البرمجة المتطورة للأجهزة المحمولة بلغة J2ME

البرمجة المتطورة للأجهزة المحمولة

بلغة

J2ME

تأليف

المهندس فهمي الصيرفي

البرمجة المتطورة للأجهزة المحمولة بلغة J2ME

تأليف: المهندس فهمي الصيرفي

الطبعة الأولى: ٢٠٠٨.

عدد النسخ: ١٠٠٠ نسخة.

جميع العمليات الفنية والطباعية تمت في:

دار ومؤسسة رسلان للطباعة والنشر والتوزيع

يطلب الكتاب على العنوان التالي

دار رسلان

للطباعة والنشر والتوزيع

سوريا - دمشق - جرمانا

هاتف: ٠٠٩٦٣ ١١ ٥٦٢٧٠٦٠

فاكس: ٠٠٩٦٣ ١١ ٥٦٣٢٨٦٠

ص. ب: ٢٥٩ جرمانا

الفصل الأول

مقدمة نظرية حول J2ME، معلومات عامه حول J2ME

إن J2ME هـي إحدى تقنيات الجافا المستخدمة لعمل تطبيقات للأجهزة الصغيرة small device.

أضيفت هذه التقنية إلى الجافا مؤخراً تقريباً في عام ١٩٩٩، وشهدت هذه التقنية تطوراً كبيراً في عام ٢٠٠٨ مع تطور الأجهزة التي تدعم J2ME

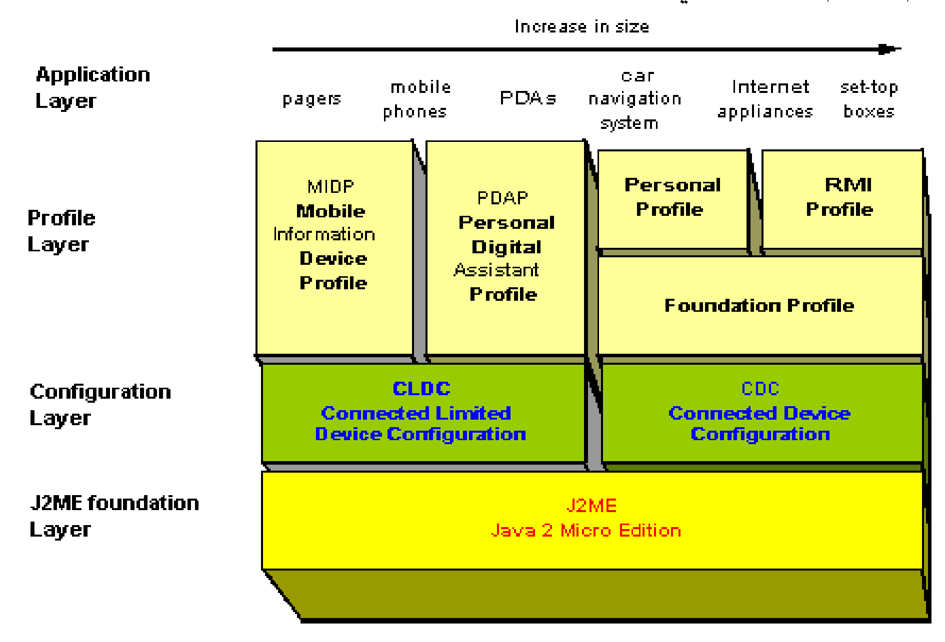
وبهذا يمكننا تقسيم الجافا إلى ثلاثة أقسام:

١- J2EE : تستخدم لبناء التطبيقات الكبيرة.

٢- J2SE : تستخدم لبناء التطبيقات التجارية المتوسطة ومنها التطبيقات التي تعمل على أجهزة الحاسب المكتبية.

٣- J2ME : تستخدم لبناء التطبيقات على الأجهزة الصغيرة مثل الموبايلات وأجهزة البوكت وغيرها.

ويتم تقسيم J2ME إلي طبقات:



١- J2ME Foundation Layer : الطبقة الأساسية J2ME.

٢- Layer Configuration : تحوي هذه الطبقة على مجموعات الأجهزة ذوات الذاكرة المحدودة والمعالج المحدود، وتنقسم إلى قسمين CLDC و CDC (Connected Limited Device Configuration و Connected Device Configuration).

٣- Profile Layer : تحوي على المجموعات التي تستخدم دوال APIs واحدة.

٤- Application Layer : التطبيق الذي تنتجه سيعمل على مجموعة من الأجهزة المحددة.

العتاد المطلوب في الأجهزة التي تدعم J2ME:

يجب أن يحتوي الجهاز على:

١ - ذاكرة كلية تترأوح بين ١٦٠KB إلى ٥١٢KB.

٢ - معالج ٣٢Bit هو ١٦Bit ويعمل بسرعة لا تقل على ٢٥MHZ.

٣ - ذاكرة غير متطايرة ١٢٨KB لحفظ المكتبات الخاصة بال CLDC.

٣ - ذاكرة متطايرة ٣٢KB لتشغيل التطبيقات.

البرمجيات المطلوبة للبدء ببرمجة J2ME:

هناك طريقتان إما العمل على Wireless Toolkits أو بالعمل على Borland JBuilder MobileSet . بالنسبة للـ Wireless Toolkits يجب عليك تنزيل j2sdk بعد ذلك تنزيل j2me_wireless_toolki .

وسوف تجدهما على هذا الرابط:

http://java.sun.com/j2me/download.html

بالنسبة لـ JBuilder MobileSet يجب أولا تحميل JBuilder بعد ذلك نقوم بتحميل JBuilder MobileSet ويجب الانتباه إلى مسألة التوافق مثلاً: j2sdk-1_3_0_02 تعمل مع j2me_wireless_toolkit-1_0_4_01 أيضاً JBuilder 6 Borland يعمل مع MobileSet 2.0

الفصل الثاني

التطبيق الأول باستخدام J2ME، خطوات بناء التطبيق

عرفنا فيما سبق أن J2ME هي القسم الثالث من أقسام الجافا بعد J2SE و J2EE.

وهذا القسم J2ME يعمل على الأجهزة الصغيرة device Small، والآن سنتعرف على المكتبات الخاصة بهذه اللغة وكذلك أنواع المعطيات المستخدمة وهيكلية بناء البرنامج، ثم ننتقل إلى بناء التطبيق الأول باستخدام J2ME.

❖ المكتبات الأساسية الخاصة بالـ J2ME هي:

javax.microedition.lcdui: تزودنا هذه المكتبة بالعديد من واجهات المستخدم مثل النوافذ والأزرار وصناديق النصوص.

javax.microedition.rms: تزودنا هذه المكتبة بالطرق المستخدمة لخزن البيانات على وحدة الخزن الرئيسية للجوال.

javax.microedition.midlet: تزودنا هذه المكتبة بال MIDP وكذلك بالبيئة التي سيعمل فيها التطبيق.

javax.microedition.io: تزودنا هذه المكتبة بالمناهج الخاصة بعمليات الاتصال GenericConnection framework GCF java.io تزودنا هذه المكتبة بالمناهج الخاصة بعمليات الإدخال والإخراج.

java.lang: تحتوي على المناهج الخاصة بال J2SE وأنواع المعطيات.

java.util: تحتوي على المناهج الملحقة.

كما إن هناك العديد من المكتبات الأخرى التي تعمل على أجهزة معينة فقط بينما هذه المكتبات هي المكتبات الأساسية الخاصة بال J2ME.

❖ أنواع المعطيات في J2ME:

CODE

```
java.lang.Boolean
java.lang.Byte
java.lang.Character
java.lang.Integer
java.lang.Long
java.lang.Short
```

ونلاحظ هنا أن J2ME لا تدعم الأعداد ذوات النقطة العائمة float , double وذلك لأن النقطة العائمة تحتاج إلى عتاد كبير وبرمجيات ذات تكلفة عالية

❖ بناء البرنامج الأول:

عند قراءة هذا الموضوع يفترض أن يكون عندك معلومات لا بأس بها عن البرمجة الكائنية.

في برنامج الجافا أو السي مثلا نرى أن بداية تنفيذ الكود يبدأ من الدالة main أما بالنسبة للـ J2ME فإن الأمر مختلف قليلاً وسيبدأ تنفيذ البرنامج في J2ME من الدالة startApp وذلك لسبب بسيط وهو:

عندما نقوم بتحديد برنامج ما هو البرنامج الرئيسي في الجافا أو السي يكفي أن نقوم بكتابة الدالة Main لنشير للمترجم أن يبدأ من هنا ، أما في J2ME فإننا نقوم بإنشاء منهج مشتق من المنهج MIDlet الموجود في الحزمة javax.microedition.midlet ، ويوجد في هذا المنهج الطرق المجردة abstract Methods الثلاث التالية , pauseApp , destroyApp , startApp .

ومن أساسيات البرمجة الكائنية أن الطرق المجردة في صنف مشتق منه يجب ذكرها كاملة ، ولهذا فإن البرنامج الرئيسي سيكون هو البرنامج الذي سيتم

اشـتقاقه مـن الـصنف MIDlet وسـيبدأ تنفيـذ البرنـامج مـن الدالـة startApp أيضاً هناك الطريقتان pauseApp , destroyApp.

pauseApp: يتم تنفيـذ هـذه الدالـة عنـد انتقـال التطبيـق مـن الحالـة النـشطة إلى الحالة الغير نشطة.

destroyApp: يتم تنفيذ هذه الدالة عند إغلاق التطبيق.

❖ **دورة حياة البرنامج الرئيسي** MIDlet

يمر التطبيق بثلاثة مراحل أو حالات:

الحالـة الابتدائيـة النـشطة ثـم الحالـة غـير النـشطة (التوقـف المؤقـت) ثـم حالـة الإغـلاق وإنهاء التطبيق.

لا تهتم حالياً بالحالة pauseApp ولكـن سنركز علـى الحـالتين startApp , destroyApp.

يجب ملاحظـة إنـه عنـد تنفيـذ أي تطبيـق فإنـه سـيقوم أولاً بتنفيـذ الكـود الموجـود في المشيد Constructor والذي يحمل نفس اسم المنهج، وبعد ذلـك سينتقل التنفيـذ إلى الدالة startApp.

❖ والآن سنقوم بكتابة المثال التالي:

قـم بكتابـة هـذا الكـود مـع ملاحظـة إنـه يجـب حفـظ هـذا الملـف بـنفس اسـم المنهج HelloMidlet

CODE

```
import javax.microedition.midlet.*;

import javax.microedition.lcdui.*;


public class HelloMidlet extends MIDlet
{
```

```
private Display display;


TextBox box = null;

public void HelloMidlet(){}

public void startApp(){


display =Display.getDisplay(this);

box =new TextBox("Simple Example ","Hello
",20,0);

display.setCurrent(box);
}

public void pauseApp(){ }

public void destroyApp(boolean unconditional){}
}
```

❖ لتحويل هذا البرنامج إلى تطبيق على الجوال سنتبع الخطوات التالية:

خطوات تحويل البرنامج المصدر (java) إلى (jad ,jar, لمن يستخدمون j2me)

١ - يقوم هذا الآمر بترجمة الملف المصدر java إلى class والتحقق من الأخطاء القواعدية للغة الجافا بنية الآمر javac.

CODE

```
C:\> %jdk1.3.0_02\bin\%javac -g:none -d
tmpclasses -bootclasspath $midpapi -classpath
$J2MECLASSPATH MyProgram.java
```

العبارات

CODE

```
javac
-g
-d
-bootclasspath
-classpath
```

هي عبارات ثابتة لا تتغير حسب تغير البرنامج، أما بقية العبارات فهي عبارات متغيرة حسب البرنامج والمجلد أو الملفات المتضمنة مكونات الأمر javac.

٪jdk1.3.0_02\bin\% : الدليل الحالي للملف javac.exe غالباً يكون اسمه jdk1.3.0_02\bin\ .

ويمكنك الاستغناء عن كتابته إذا قمت بكتابة الآمر من داخل الدليل للملف javac.exe .

none: تعني عدم إظهار التنقيح.

وهذا الخيار غير مهم ويمكن الاستغناء عنه وعدم كتابته.

tmpclasses : المجلد الذي ستوضع فيه الملفات class بعد ترجمتها.

أيضاً هذا الخيار غير مهم ويمكن الاستغناء عنه وسيتم وضع الملفات .class في المجلد الحالي ٪jdk1.3.0_02\bin\٪ للملف javac.exe .

$midpapi : هـذا هـو package أو الكلاسـات الأساسـية لـل J2ME وتكـون مـــضغوطة علـــى هيئـــة ملـــف .zip وغالبـــاً مـــا تكـــون في الـــدليل C:\WTK104\lib\midpapi.zip .

$J2MECLASSPATH : الكلاســات الإضــافية الــتي قمــت بتــضمينها في برنامجك.

مـثلاً لـو كـان لـديك أكثـر مـن كـلاس في برنـامج واحـد يجـب عليـك تحـزيم الكلاسات الإضافية في حزمة واحدة، وبعد ذلك ستقوم بكتابة اسم هذه الحزمة مع المسار بدلاً عن الكلمة $J2MECLASSPATH وغالباً يتم تضمين هذه الكلاسات إلى الحزمة C:\WTK104\wtklib\kvem.jar .

مثال على ذلك لنفترض أنه لدينا البرنامج HelloMidlet والموجود في المجلد C:\java ونريد ترجمته بالعبارة javac وإرسال الملفات الناتجة .class إلى الدليل C:\WTK104\bin\Classes\ .

سنقوم بكتابة الآمر :

CODE

```
C:\jdk1.3.0_02\bin\javac.exe -g:none -d
C:\WTK104\bin\Classes\ -bootclasspath
C:\WTK104\lib\midpapi.zip -classpath
C:\WTK104\wtklib\kvem.jar
C:\Java\HelloMidlet.java
```

يجب الانتباه إلى كتابة الأوامر بالحروف الصغيرة أو الكبيرة تبعاً لأسماء الملفات مثلاً لو قمنا بكتابة -bootclasspath C:\WTK104\lib\MIDPAPI.zip فإنه ستنتج لدينا عبارة خطأ لأن الملف MIDPAPI.zip مكتوب بحروف كبيرة بينما هو في الأصل مكتوب بحروف صغيرة.

قد تدور بذهنك هذه الأسئلة:

ماذا لو كان لدينا أكثر من ملف ونريد ترجمة كل هذه الملفات؟

كما ذكرنا سابقاً عندما شرحنا J2MECLASSPATH$ نقوم أولاً بترجمة الكلاس المستدعى الإضافي -وليس الذي يستدعي- ونمر بالمراحل الثلاث إلى أن نصل إلى الشكل .jar وبعد ذلك نقوم بوضع اسمه مع الدليل بدلاً عن المتغير J2MECLASSPATH$.

والآن سيدور بذهنك سؤال آخر ماذا لو كان كل كلاس يقوم باستدعاء الكلاس الآخر؟

أي أن الكلاسات تستدعي كلاسات أخرى وهذه الكلاسات الأخرى تستدعي نفس الكلاسات الأصلية، وحل هذه المسألة هو أيضاً حل آخر للمسألة السابقة -أي حتى لو كانت الكلاسات لا تستدعي بعضها البعض- وهو حل في غاية البساطة فقط سوف تقوم بذكر أسماء الملفات .java كلها في عبارة واحدة.

مثال لو كان لدينا الملفات test1.java و test2.java وكان هذان الكلاسان يستدعيان بعضهم البعض أو أحدهم يستدعي الآخر فإنه يمكن كتابة الأمر javac على الشكل:

CODE

```
C:\jdk1.3.0_02\bin\javac.exe -g:none -d
C:\WTK104\bin\Classes\ -bootclasspath
C:\WTK104\lib\midpapi.zip -classpath
C:\WTK104\wtklib\kvem.jar test1.java test2.java
```

٢ - المرحلة التالية مرحلة التحقق من byte code باستخدام الأمر perversity ويتم في هذه المرحلة التحقق من وجود الكلاسات الأخرى في البرنامج وسوف يتم إضافة مجموعة من البيانات على الكلاس .class ويمكن معرفة ذلك من خلال مراقبة حجم الكلاس قبل تنفيذ هذه المرحلة وبعد تنفيذ هذه المرحلة، وبنية الأمر كالتالي:

CODE

```
C:\>%WTK104\bin\% perversity -class path
$midpapi;tmpclasses -d classes tmpclasses
```

العبارات

CODE

```
perversity
-class path
;
-d
```

هي عبارات ثابتة لا تتغير حسب تغير البرنامج، أما بقية العبارات فهي عبارات متغيرة حسب البرنامج والمجلد أو الملفات المتضمنة.

%WTK104\bin\%: الدليل الحالي للملف preverify.exe غالباً يكون اسمه WTK104\bin\ ويمكنك الاستغناء عن كتابته إذا قمت بكتابة الأمر من داخل المجلد.

$midpapi: هو نفسه الملف المشروح في الخطوة السابقة عبارة عن جميع الكلاسات الخاصة باللغة J2ME.

tmpclasses : هو نفسه المجلد المشروح في الفقرة السابقة الدليل التي تتواجد فيه الكلاسات .class بعد تحويلها من الشفرة المصدرية إلى byte code وسيتم التحقق من كل الكلاسات الموجودة في هذا الدليل ولن نحدد كلاس واحد .class .

classes: هذا الدليل أو المجلد هو المجلد الذي سيوضع فيه الملفات .class بعد التحقق منها.

مثال على ذلك: لنفرض أن الكلاس الذي قمنا بترجمته في الخطوة السابقة قد قمنا بوضعه في الدليل C:\WTK104\bin\Classes\ .

والآن نريد استخدام الأمر perversity ووضع الكلاسات الناتجة من هذا الأمر في المجلد \C:\jdk1.3.0_02\bin فإننا سنقوم بكتابة الأمر التالي:

CODE

```
C:\WTK104\bin\preverify.exe -class path
C:\WTK104\lib\midpapi.zip;C:\WTK104\bin\Classes\
-d C:\jdk1.3.0_02\bin\ C:\WTK104\bin\Classes\
```

ويجب أن تلاحظ هنا أن هذا الأمر يقوم بالتنفيذ على عدد من الكلاسات موجودة في مجلد واحد وليس على كلاس واحد بعينه مثلاً لو كان لدينا الكلاسات test1.class , test2.class الموجودين في الدليل C:\WTK104\bin\Classes\ .

سنقوم بكتابة نفس الأمر السابق لجميع الكلاسات test2.class test1.class .

CODE

```
C:\WTK104\bin\preverify.exe -class path
C:\WTK104\lib\midpapi.zip;C:\WTK104\bin\Classes\
-d C:\jdk1.3.0_02\bin\ C:\WTK104\bin\Classes\
```

٣ - تحزيم الملفات باستخدام الأمر jar:

يقوم هذا الأمر بضغط الكلاسات وتجميعها في حزمة واحدة يتم تنفيذها على الموبايل، وبنية هذا الأمر كالتالي:

CODE

```
%C:\jdk1.3.0_02\bin%> jar cvf MyProgram.jar
MyProgram.class
```

أو على الطريقة التالية:

CODE

```
%C:\jdk1.3.0_02\bin%> jar cmf MANIFEST.MF
MyProgram.jar MyProgram.class
```

والفرق بين الطريقتين هي أن في الطريقة الأولى لا نقوم بتضمين الملف MANIFEST.MF وهذا الملف مهم جداً على بعض الأجهزة مثل نوكيا وليس مهما على أجهزة أخرى مثل سيمنس بينما في الطريقة الثانية سنقوم بتضمين الملف MANIFEST.MF الذي سنقوم بشرح كيفية تكوينه فيما بعد.

ويجب الانتباه هنا إلى أننا قمنا بالدخول إلى الدليل أولاً قبل استخدام الأمر jar على العكس من المراحل السابقة التي لا تشترط ذلك. والدخول إلى الدليل مهم جداً من أجل أن يتم تحزيم الكلاسات .class في الدليل الرئيسي للملف jar .

أما في حالة عدم الدخول للدليل المحتوى على الأمر jar فإن تحزيم الكلاسات في الملف jar سيكون داخل الملف الفرعي الموجود فيه الكلاسات التي تم تحزيمها.

لا تهتم إذا لم تفهم ما ذكر سابقاً فقط قم بالدخول إلى الدليل المحتوي على الأمر jar واستخدم الأمر من داخل المجلد.

%C:\jdk1.3.0_02\bin: الدليل الحالي للملف jar.exe .

cvf أو cmf أو umf: يعبر عن الطريقة المستخدمة في تنفيذ الأمر، وسنقوم بشرحها لاحقاً.

MyProgram.jar: اسم البرنامج .jar الذي تريد تحزيمه.

MyProgram.class: اسم الكلاس أو الكلاسات التي تريد تحزيمها.

مثال سنقوم بتحزيم المثال السابق بالشكل:

CODE

```
C:\jdk1.3.0_02\bin\jar.exe cvf Hello.jar
 HelloMidlet.class
```

سيتكون لدينا البرنامج Hello.jar الناتج من تطبيق الأمر السابق.

والآن سيتبادر للذهن ماذا لو كان لدينا أكثر من كلاس واحد ونريد تحزيمها في حزمة واحدة؟

مثلاً الكلاسات test2.class test1.class ونريد تحزيمها إلى test.jar سنقوم باستخدام الأمر التالي:

CODE

```
C:\jdk1.3.0_02\bin\jar.exe cvf test.jar  test1.class
test2.class
```

يجب أن نذكر هنا أن استخدام البارامتر umf يستخدم لإضافة الملحقات المستخدمة في البرنامج مثل الصور أو أيقونات التطبيق.

مثلاً لو أردنا إضافة الصورة test.png إلى البرنامج Hello.jar طبعاً هذا إذا كنا قد استخدمنا هذه الصورة داخل الكلاس HelloMidlet.class في شفرة البرنامج، سوف نستخدم الأمر التالي:

CODE

```
C:\jdk1.3.0_02\bin\jar.exe umf MANIFEST.MF
Hello.jar Test.png
```

٤ - يتم إنشاء وتكوين ملفات jad و MANIFEST.MF باستخدام أي محرر للنصوص، وتتم كتابة عدة حقول فيهما منها ما هو ضروري ومنها ما هو ليس ضروري.

وهذا الجدول المرفق يشرح هذه الحقول:

يمكنك فتح أي ملفات jad لبرامج أخرى لمعرفة كيفية كتابتها

JAR manifest attributes		
اسم الخاصية	مطلوب	شرح الخاصية
MicroEdition-Configuration	نعم	رقم النسخة من J2ME Configuration مثلاً CLDC-1.0
MicroEdition-Profile	نعم	رقم النسخة من J2ME Profile مثلاً MIDP-1.0
MIDlet-n	نعم	اسم التطبيق, اسم الأيقونة , اسم الكلاس الرئيسي والرقم n- يعبر عن رقم التطبيق تبدأ بـ ١
Data-Size-MIDlet	لا	العدد الأقصى من البايتات الذي سيستخدمه التطبيق في التعامل مع الملفات
MIDlet-Description	لا	يحتوي على وصف للتطبيق
Icon-MIDlet	لا	المسار مع الملف لأيقونة التطبيق
MIDlet-Info-URL	نعم	عنوان الإنترنت الذي يحتوي على وصف للتطبيق
MIDlet-Name	نعم	اسم التطبيق
MIDlet-Vendor	نعم	معلومات منتج البرنامج
MIDlet-Version	نعم	رقم نسخة التطبيق

JAD attributes		
شرح الخاصية	مطلوب	اسم الخاصية
اسم التطبيق	نعم	MIDlet-Name
رقم نسخة التطبيق	نعم	MIDlet-Version
معلومات منتج البرنامج	نعم	MIDlet-Vendor
اسم الكلاسات المحزمة Jar.	نعم	MIDlet-Jar-URL
حجم الملف - jar. - بالبايتات	نعم	MIDlet-Jar-Size
يحتوي على وصف للتطبيق	لا	MIDlet-Description
المسار مع الملف لأيقونة التطبيق	لا	Icon-MIDlet
عنوان الإنترنت الذي يحتوي على وصف للتطبيق	نعم	MIDlet-Info-URL
العدد الأقصى من البايتات الذي سيستخدمه التطبيق في التعامل مع الملفات	لا	Data-Size-MIDlet

وهنا يجب ملاحظة أن البيانات الموجودة في الملف MANIFEST.MF والبيانات الموجودة في الملف jad يجب أن تتطابق وإلا فإن التطبيق لن يعمل في بعض الأجهزة مثل أجهزة نوكيا.

وسوف تظهر للمستخدم الرسالة التالية : صيغة الملف غير مدعومة.

والآن أصبح لدينا برنامج جاهز للعمل على أي موبايل.

لتجربة هذا التطبيق قبل تنزيله على الموبايل:

قم أولاً بنقل التطبيق إلى الدليل apps الموجود في الدليل الخاص بال J2ME وليكن C:\WTK104 أي C:\WTK104\apps.

بعد ذلك قم بفتح الملف jad باستخدام emulatorw.exe الموجود في الدليل الخاص بال J2ME وليكن C:\WTK104 أي C:\WTK104\bin .

تعتبر هذه المراحل في ترجمة البرنامج مملة وكبيرة ومعقدة ولكنها مهمة في بداية تعلم اللغة لمعرفة المراحل التي يمر بها البرنامج.

للسهولة يمكنك عمل قوالب جاهزة بملفات batche file.bat ومن ثم تغييرها عند تغيير اسم البرنامج

الفصل الثالث

إضافة الأزرار (commands) إلى البرنامج في J2ME، كيفية بناء التطبيق مع استخدام Commans

بعد أن تعرفنا على كيفية بناء التطبيق الأول وكيفية تحويل هذا الكود إلى تطبيق يعمل على الجوال سنقوم الآن بشرح لكيفية إضافة الأزار والتعامل معها بعد هذا الشرح يفترض بك أن تكون قادر على بناء تطبيق صغير من تلقاء نفسك يجب عليك أولا إنشاء زر جديد ، ثم إضافة هذا الزر إلى الواجهة التي تريد عرضها ، بعد ذلك ستضع الكود الذي تود تنفيذه عند الضغط على هذا الزر.

١ - تعريف الأزرار في البرنامج:

أولاً : سنقوم بتعريف كائن جديد من نوع Command

CODE

```
Command command;
```

٢ - ثم نقوم بإنشاء هذا الكائن باستخدام الأمر new ونقوم بإعطاء الزر بعض الخصائص مثل الاسم الذي سيظهر للمستخدم وكذلك نوع هذا الزر وأولوية الظهور

CODE

```
command = new Command(String label, int
commandType, int priority)
```

ويمكن اختصار الطريقتين السابقتين على الشكل:

CODE

```
Command command = new Command(String
label, int commandType, int priority)
```

أنشأنا كائن من نوع زر ثم أعطيناه بعض الخصائص.

٣ - ربط هذا الزر مع واجهة محددة بعد ذلك نقوم بإضافة هذا الزر إلى الواجهة UI التي نريد عرض الزر معها باستخدام الطريقة:

CODE

```
UI.addCommand(command);
```

مثلاً لو كانت الواجهة TextBox box سنكتب الأمر على الشكل:

CODE

```
box.addCommand(command);
```

٤ - إضافة الكود الذي سينفذ عند الضغط على الزر من أجل إضافة كود لزر أمر معين يجب علينا استخدام implements Interface .
أي إننا سنضمن الواجهة CommandListener في الكلاس الذي سنستخدم فيه الأزرار بهذا الشكل.

CODE

```
public class HelloMidlet extends MIDlet implements
CommandListener  {
```

يوجد في الواجهة CommandListener الطريقة المجردة commandAction(Command c, Displayable s) لهذا يجب تضمين هذه الطريقة في تطبيقنا لأننا استخدمنا implements CommandListener .

تعيد لنا هذه الواجهة كائنان الأول Command c وهو الزر الذي قمنا بضغطه الثاني Displayable s وهي الواجهة UI الذي تحوي هذا الزر تعمل هذه الواجهة على تشغيل المتنصت Listener لكي يتصنت على كبسة الزر يجب أولاً توجيه هذا المتنصت إلى UI محددة من خلال الأمر setCommandListener أي إننا سنخبر المتنصت أن ينفذ الكود عند حدوث كبسة زر ما من تلك الواجهة UI .

والآن سنضع هذا المثال المطور عن مثالنا السابق ونعود بعد ذلك لمناقشة الأشياء الجديدة فيه.

CODE

```
import javax.microedition.midlet.*;

import javax.microedition.lcdui.*;


public class HelloMidlet extends MIDlet implements
CommandListener
{

private Display display;
Command command = new Command("Exit",
Command.EXIT, 0);

TextBox box;

public void HelloMidlet(){}

public void startApp(){


display =Display.getDisplay(this);

box =new TextBox("Simple Example ","Hello
",20,0);
box.addCommand(command);
box.setCommandListener(this);
display.setCurrent(box);
}
```

```
public void pauseApp(){ }

public void destroyApp(boolean unconditional){}

/************* implements CommandListener
****************/

public void commandAction(Command c,
Displayable s)
 {
   if ( c == command && s == box )
     {
       destroyApp(false);
       notifyDestroyed();
     }
 }

}
```

CODE

```
public class HelloMidlet extends MIDlet implements
CommandListener
```

implements : يعني أن هذا الكلاس سيستخدم واجهة أو أكثر من واجهة تفصل بينهما فاصلة صغيرة، Interface1,Interface2 .

CommandListener : عبارة عن واجهة تستخدم لإضافة رد فعل معين تجاه كبسة زر معين.

CODE

```
Command command = new Command("Exit",
Command.EXIT, 0);
```

قمنا بإنشاء الكائن command والذي سيظهر للمستخدم بالاسم Exit.

CODE

```
box =new TextBox("Simple Example ","Hello
",20,0);
```

إنشاء كائن من نوع صندوق نص TextBox عنوان هذا الصندوق Simple Example .

وعدد الحروف التي يسمح بها ٢٠ حرف، أما القيمة ٠ فتمثل TextField.ANY وهذا يعني أن صندوق النص سيتقبل أي حرف أو رقم.

CODE

```
box.addCommand(command);
```

قمنا بإضافة الزر الذي أنشأناه سابقاً إلى box باستخدام الطريقة addCommand الموجودة في جميع الواجهات UI.

CODE

```
box.setCommandListener(this);
```

أخبرنا المتنصت بأن يقوم بالتنصت على الواجهة box باستخدام الأمر setCommandListener الموجود في جميع الواجهات UI أما الكلمة this فتعني الكلاس الحالي الذي نحن فيه، وهذا يعني أننا سوف نقوم بمعالجة كبسة الزر في هذا الكلاس.

CODE

```
public void commandAction(Command c,
Displayable s)
```

هذه هي الطريقة التي تعمل مع الواجهة CommandListener وتعيد بارامتران الأول هو الزر الذي تم كبسه والثاني الواجهة UI التي تحوي هذا الزر.

CODE

```
if ( c == command && s == box )
```

التحقق من أن الزر الذي تم كبسه والتأكد من الواجهة التي تحوي هذا الزر.

CODE

```
        destroyApp(false);
        notifyDestroyed();
```

هذان الأمران يقومان بإغلاق التطبيق. الأول يقوم بتشغيل عملية الإغلاق باستدعاء الدالة destroyApp والثاني يقوم بالتحسس لعملية الإغلاق.

الفصل الرابع

التعامل مع الواجهات High Level APIs في J2ME، مقدمة نظرية

بعد أن تعرفنا على كيفية بناء التطبيق وكذلك تعرفنا على كيفية إضافة الأزرار Commands إلى التطبيق، سنتعرف الآن على بقية الواجهات التي تستخدم للعرض مثل TextBox , List , Form ,Canvas , Graphics .

ويتم تقسيم هذه الواجهات إلى قسمين:

١ - high-level API .

٢ - low-level API .

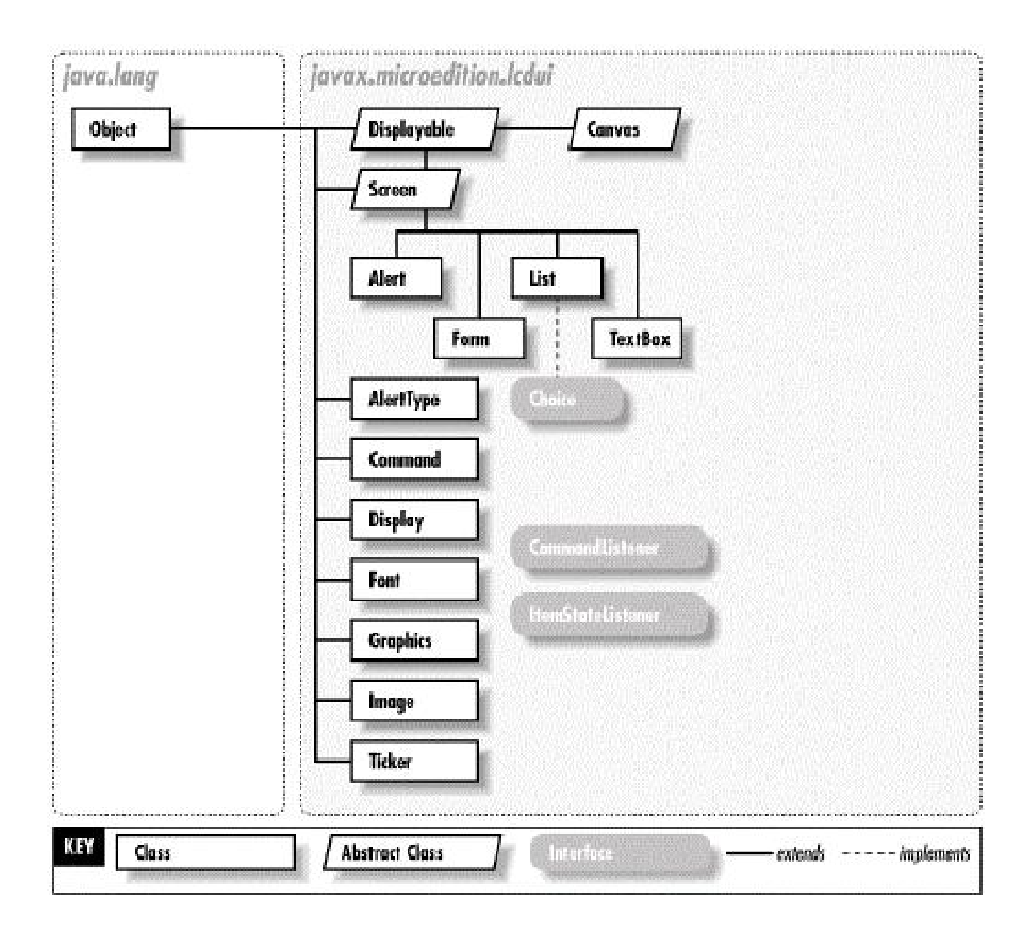
تستخدم الأولى النمط الحرفي للعرض، بينما تستخدم الثانية النمط الرسومي Graphic

وكل هذه الأصناف موجودة في حزمة واحدة لل J2ME وهي:

CODE

```
Package javax.microedition.lcdui
```

ويمكنك التعرف على هيكلية بناء هذه المكتبة بالنظر إلى الصورة



بالنظر إلى الهيكلية سنتعرف على الأصناف Classes والواجهات Interfaces.

توجد ثلاث واجهات في هذه المكتبة:

Choice: ويستخدمها الصنف List فقط.

CommandListener: وقد تحدثنا عنه سابقاً عند إضافة الأزرار.

Commands: للتطبيق .

تستخدم للتنصت على كبسة زر معين.

ItemStateListener: ويستخدمها الصنف Form فقط.

بالإضافة إلى وجود العديد من الأصناف الذي سنتعرف على بعض منها لاحقاً.

الفصل الخامس

التعامل مع صندوق النص TextBox في J2ME، ضمن High Level APIs

سنتعرف في هذا الجزء على كيفية التعامل مع صندوق النص TextBox وهو عبارة عن شاشة يستطيع المستخدم من خلالها الإضافة أو التحرير المشيد Constructor الخاص بهذا الكائن على الشكل:

CODE

```
TextBox(String title, String text, int maxSize, int constraints)
```

title: عنوان النص وسيظهر في أعلى صندوق النص.

ونستطيع تغييره باستخدام الطريقة الموجودة في نفس الصنف setTitle وهذه الطريقة ورثها هذا الصنف TextBox من الصنف Screen.

text: وهو النص الذي سيكون موجوداً في داخل صندوق النص، ونستطيع التعامل معه من خلال الأمرين setString , getString .

maxSize: عدد الأحرف التي سيتقبلها صندوق النص بحيث لا يمكن للمستخدم إضافة أكثر من العدد المطلوب.

constraints: القيد أو الهيئة للنص المدخل وهناك العديد من القيود هذه القيود موجودة في الصنف TextField على هيئة حقول ساكنه static ويتم استخدامها على الشكل TextField.Field حيث Field تعبر عن اسم القيد.

من هذه القيود:

ANY: ويعبر عن جميع الأحرف أو الأرقام.

NUMERIC: يسمح للمستخدم باستخدام الأرقام فقط.

PASSWORD: تظهر للمستخدم نجمة عن كل حرف يكتبه.

وبعد أن تعرفنا على بنية المشيد سنذكر الآن بعض الطرق التابعة للصنف TextBox:

CODE

```
public void delete(int offset, int length)
```

تقوم هذه الدالة بحذف جزء من النص الموجود في صندوق النص.

offset: بداية النص المراد حذفه.

length: طول النص.

CODE

```
public int getConstraints()
```

تعيد نوع القيد المستخدم على شكل رقم.

CODE

```
public void setConstraints(int constraints)
```

تستطيع تغيير القيد المستخدم في الصندوق باستخدام هذا المنهج.

CODE

```
public int getMaxSize()
```

تعيد الحد الأقصى لعدد الحروف الممكن كتابتها في صندوق النص.

CODE

```
public int setMaxSize(int maxSize)
```

أيضاً نستطيع تغيير الحد الأقصى لعدد الحروف باستخدام هذا المنهج.

CODE

```
public String getString()
```

تعيد هذه الدالة النص المكتوب في الصندوق.

CODE

```
public void setString(String text)
```

يقوم هذا المنهج بوضع النص التي تريده على الصندوق.

CODE

```
public int size()
```

تعيد هذه الدالة عدد الأحرف المكتوبة حالياً في الصندوق.

والآن بعد شرحنا كيفية التعامل مع صندوق النص بقي علينا أن نقوم بوضع مثال يوضح لنا كل هذا:

CODE

```
import javax.microedition.midlet.*;

import javax.microedition.lcdui.*;


public class HelloMidlet extends MIDlet implements
CommandListener
{

private Display display;
Command cmdExit = new Command("Exit",
```

```
Command.EXIT, 0);
Command cmdClear = new Command("Clear",
Command.SCREEN, 0);
Command cmdUndo = new Command("Undo",
Command.SCREEN, 0);
Command cmdConstraint = new
Command("Constraint", Command.SCREEN, 0);

TextBox box = null;
String undoString;

public void HelloMidlet(){}

public void startApp(){


display =Display.getDisplay(this);

undoString = "Hello";
box =new TextBox("ANY
",undoString,20,TextField.ANY);
box.addCommand(cmdExit);
box.addCommand(cmdClear);
box.addCommand(cmdUndo);
box.addCommand(cmdConstraint);
box.setCommandListener(this);
display.setCurrent(box);
}

public void pauseApp(){ }

public void destroyApp(boolean unconditional){}
```

```
/************* implements CommandListener
****************/

public void commandAction(Command c,
Displayable s)
 {
   if (s == box)
    {
     if ( c == cmdExit )
       {
        destroyApp(false);
        notifyDestroyed();
       }
     else if ( c == cmdClear )
      {
        undoString = box.getString();
        box.setString("");
      }
     else if ( c == cmdConstraint )
        if ( box.getConstraints() == TextField.ANY)
          {
           box.setConstraints(TextField.NUMERIC);
           box.setTitle("NUMERIC");
          }
        else
          {
           box.setConstraints(TextField.ANY);
           box.setTitle("ANY");
          }
     else if ( c == cmdUndo )
        box.setString(undoString);
```

```
    }

 }  //  end commandAction

}  // end Class
```

CODE

```
Command cmdExit = new
Command("Exit",
Command.EXIT, 0);
```

سنستخدمه من أجل الخروج من البرنامج.

CODE

```
Command cmdClear = new Command("Clear",
Command.SCREEN, 0);
```

سنستخدمه من أجل مسح محتويات صندوق النص.

CODE

```
Command cmdUndo = new Command("Undo",
Command.SCREEN, 0);
```

سنستخدمه من أجل عرض النص السابق قبل عملية المسح الأخيرة.

CODE

```
Command cmdConstraint = new
Command("Constraint", Command.SCREEN, 0);
```

سنستخدمه من أجل تغيير القيد في صندوق النص.

عرفنـا ثلاثـة أزرار مـن النـوع SCREEN و زر واحـد مـن النـوع EXIT طبعـاً هـذه الأنواع SCREEN , EXIT ليست مهمـة جداً ولكـن لترتيـب عمليـة الظهـور علـى الشاشة، مثلاً إذا قمنـا بتعريف زر معين من نـوع BACK فإن هـذا الـزر سيظهر في يمين الشاشة على العكس من النوع SCREEN الـذي سيظهر في اليمين، طبعـاً تختلف طريقة الظهور من جهاز لآخر .

CODE

```
String undoString;
```

هنا عرفنا متغير من نوع نصي لنقوم بخزن النص الموجود في صندوق النص قبل حذفه في هذا المثال، وذلك حتى يستطيع المستخدم العودة للنص السابق.

CODE

```
undoString = "Hello";
```

اسـندنا القيمـة النـصية Hello إلى المـتغير undoString وذلك حتى نقوم بعرض هذه القيمة النصية وفي نفس الوقت نحتفظ بالنص السابق.

CODE

```
box =new TextBox("ANY
",undoString,20,TextField.ANY);
```

عنـوان صندوق النص هـذا سيحمل اسـم القيـد Constraint وفي البداية سيكون القيد من النوع ANY.

CODE

```
box.addCommand(cmdExit);
box.addCommand(cmdClear);
box.addCommand(cmdUndo);
box.addCommand(cmdConstraint);
box.setCommandListener(this);
```

قمنا بإضافة جميع الأزرار المعرفة إلى صندوق النص.

CODE

```
if (s == box)
```

للتأكد من الزر الذي تم كبسه يتبع صندوق النص box وسيدخل التنفيذ هنا إذا تحقق هذا الشرط.

طبعاً في مثالنا هذا يمكن الاستغناء عن هذا الشرط لأنه لا يوجد لدينا سوى واجهة عرض UI واحدة وهي box

CODE

```
if ( c == cmdExit )
```

سيدخل التنفيذ هنا إذا تم ضغط زر الخروج Exit.

CODE

```
else if ( c == cmdClear )
```

سيدخل التنفيذ هنا إذا تم ضغط زر مسح صندوق النص Clear.

CODE

```
undoString = box.getString();
```

قبل أن نقوم بمسح محتويات صندوق النص سنأخذ النص المكتوب فيه .

CODE

```
box.setString("");
```

لمسح محتويات صندوق النص سنقوم بتعبئة الصندوق بالفراغ.

CODE

```
else if ( c == cmdConstraint )
```

سيدخل التنفيذ هنا إذا تم ضغط زر تغيير القيد Constraint

CODE

```
if ( box.getConstraints() == TextField.ANY)
```

إذا كان القيد السابق من النوع ANY سنقوم بتغييره إلى النوع NUMERIC والعكس كذلك إذا كان من النوع NUMERIC سنحوله إلى ANY .

CODE

```
box.setConstraints(TextField.NUMERIC);
```

قمنا بتغيير القيد إلى NUMERIC .

CODE

```
box.setTitle("NUMERIC");
```

غيرنا عنوان صندوق النص إلى اسم القيد الجديد NUMERIC .

CODE

```
box.setConstraints(TextField.ANY);
```

قمنا بتغيير القيد إلى ANY .

CODE

```
box.setTitle("ANY");
```

غيرنا عنوان صندوق النص إلى اسم القيد الجديد ANY.

CODE

```
else if ( c == cmdUndo )
```

سيدخل التنفيذ هنا إذا تم ضغط زر تراجع Undo.

CODE

```
box.setString(undoString);
```

فقط سوف نقوم بعرض النص الذي قمنا بحفظه في خطوة سابقة.

الفصل السادس

التعامل مع الفورم Form في J2ME، ضمن High Level APIs

سنتعرف في هذا الجزء على كيفية التعامل مع الفورم Form .

يرث هذا الصنف الكثير من الأعضاء Members من الصنف Screen ومن هذه الطرق:

CODE

```
setTicker, setTitle
```

Form: عبارة عن شاشة شبيهة إلى حد ما بـ Label حيث لا يستطيع المستخدم تحريرها.

ونستطيع إضافة الكثير من العناصر Items إلى الفورم.

سنتعرف على بعض هذه العناصر لاحقاً.

هذا الكائن يحتوي على مشيدين Constructor تستطيع اختيار أحدهما

CODE

```
public Form(String title)
public Form(String title, Item[] items)
```

title: العنوان الذي سيظهر أعلى النافذة ونستطيع تغييره باستخدام الطريقة الموجودة في نفس الصنف setTitle وهذه الطريقة ورثها الصنف Form من الصنف Screen.

items: مصفوفة من العناصر التي نود أضافتها للفورم ونستطيع التحكم بإضافة أو حذف هذه العناصر باستخدام عدد من الطرق التي سنشرح بعض منها لاحقاً

وقبل أن نقوم بشرح بعض الطرق التابعة Append , Insert , get , delete لهذا الكائن سنقدم شرحا مختصراً للعناصر Items وكذلك للكائن Ticker.

العناصر Items: هي الأصناف المشتقة من الصنف Item وهذه الكائنات هي StringItem, ChoiceGroup, DateField, Gauge, ImageItem, TextField.

وسنتعرف على بعض منها لاحقاً بإذن الله وفي هذا الفصل سنقوم بإضافة حقل النص TextField إلى الفورم وهذا الصنف شبيه إلى حد ما بالصنف TextBox أيضاً سنقوم بإضافة Ticker وهو ليس Items ولكنا سنتعرف عليه وكيف يعمل.

Ticker: عبارة عن نص يظهر في أعلى النافذة يتحرك بشكل تلقائي ويتم إضافته إلى النافذة باستخدام الطريقة setTicker التابعة للصنف Screen وبعد أن تعرفنا على بنية المشيد وما هي Items سنذكر الآن بعض الطرق التابعة للصنف Form .

CODE

```
public int append(Image img)
```

تقوم هذه الطريقة بإضافة صورة إلى الفورم بعد أن نقوم بإنشائها باستخدام الكائن ImageItems.

وسنوضح التعامل مع هذه الطريقة عندما نقوم بشرح العناصر Items.

CODE

```
public int append(Item item)
```

تقوم هذه الطريقة بإضافة عنصر ما إلى النافذة وتعيد رقم العنصر في النافذة.

CODE

```
public int append(String str)
```

إضافة أي عبارة نصية إلى الفورم باستخدام هذه الطريقة.

CODE

```
public void insert(int itemNum, Item item)
```

استبدال عنصر موجود من سابق بعنصر جديد نقوم بإضافته.

itemNum: رقم العنصر الذي تريد استبداله

item: العنصر الجديد الذي سنضعه بدلاً عن القديم.

CODE

```
public void setTicker(Ticker ticker)
```

يقوم بإضافة Ticker إلى أعلى الفورم.

وسنقوم بإنشاء الكائن Ticker باستخدام المشيد Constructor.

CODE

```
Ticker(String str)
```

حيث أن str هو النص الذي سيظهر في Ticker أعلى النافذة.

والآن بعد شرحنا كيفية التعامل مع الفورم بقي علينا أن نقوم بوضع مثال يوضح لنا كل هذا:

CODE

```
import javax.microedition.midlet.*;
import javax.microedition.lcdui.*;
```

```
public class MyForm extends MIDlet implements
CommandListener
{
 private Display display;

Command cmdExit = new Command("Exit",
Command.EXIT, 0);
 Command cmdClear = new Command("Clear
Text", Command.SCREEN, 0);

 TextField field;

  Form form;

  Ticker ticker;

 public MyForm  ()
  {
   display =Display.getDisplay(this);
   form = new Form("Form Title");
   field =new TextField("TextField Title","Arab
Team",20,0);
   ticker = new Ticker (" this text in Ticker ");
  }

 public void startApp()
  {
   form.append("This text in Fom\n");
   form.append("___________\n");
   form.append(field);
   form.addCommand(cmdExit);
   form.addCommand(cmdClear);
```

```
  form.setCommandListener(this);
  display.setCurrent(form);
 }

 public void pauseApp(){ }
 public void destroyApp(boolean unconditional){}

/************** implements CommandListener
****************/
public void commandAction(Command c,
Displayable s)
{
  if (  s == form )
    {
      if (c == cmdExit)
        {
          destroyApp(false);
          notifyDestroyed();
        }
      else if (c == cmdClear)
              field.setString("");
      }
 }  // end commandAction
} // end Class
```

الفصل السابع

التعامل مع صناديق الحوار Alert في J2ME، ضمن High Level APIs

سنتعرف في هذا الجزء على كيفية التعامل مع صناديق الرسائل أو التنبيه Alert يرث هذا الصنف عدد من الأعضاء Members من الصنف Screen ومن هذه الطرق:

CODE

```
setTicker, setTitle
```

Alert: عبارة عن شاشة شبيهة إلى حد ما بـ Message Box تظهر للمستخدم كتنبيه على عملية معينه.

عمل هذه الكائنات بسيط جداً ما عليك إلا إعطاء هذه الكائنات بعض الخصائص ومن ثم استدعائها لعرضها على الشاشة.

هذا الكائن يحتوي على مشيدين Constructor تستطيع اختيار أحدهما

CODE

```
public Alert(String title)
Alert(String title, String alertText, Image
alertImage, AlertType alertType)
```

title: العنوان الذي سيظهر أعلى الصندوق.

alertText: النص الذي سيظهر داخل التنبيه ويقرأه المستخدم.

alertImage: الأيقونة أو الصورة الصغيرة التي ستظهر في أعلى التنبيه. ويمكن كتابة null لعدم عرض أي صورة.

alertType: نوع هذا التنبيه هل هو ALARM أو CONFIRMATION أو ERROR أو INFO أو WARNING .

وكل نوع من هذه الأنواع تعطي التنبيه شكل معين ليظهر به.

ويتم كتابة النوع على الشكل AlertType.TYPE حيث نقوم باستبدال الكلمة TYPE بأحد الأنواع السابقة.

عد أن تعرفنا على بنية المشيد سنذكر الآن بعض الطرق التابعة للصنف Alert وكيفية استخدامها:

CODE

```
public void addCommand(Command cmd)
```

إضافة زر إلى التنبيه وهذه الطريقة غير مستخدمة عادة لكن قد نضطر لاستخدامها أحياناً، والسبب إنك عادة لا ترغب بإضافة أي زر عدا زر موافق. طبعاً الزر موافق سيضاف تلقائياً إلى التنبيه عند استخدام Timeout من النوع FOREVER وسنرى في الفقرات اللاحقة كيفية التعامل مع Timeout باستخدام الدالة setTimeout.

CODE

```
public void setString(String str)
```

تقوم هذه الدالة بتغيير النص الذي سيعرض في التنبيه.

CODE

```
public void setTimeout(int time)
```

Timeout: هو الوقت الذي سيبقى فيه التنبيه معروضاً على الشاشة ويقاس بالمللي ثانية.

مثلاً لو قمنا بإنشاء كائن alert من نوع الصنف تنبيه ثم أردنا عرض هذا التنبيه على الشاشة لمدة ٥ ثواني فإننا سنقوم بالكتابة كالتالي:

CODE

```
alert.setTimeout(5000);
```

لكن ماذا وإذا أردنا أن يستمر العرض إلى أن يقوم المستخدم بضغط زر موافق مثلاً.
هل سنزيد قيمة Timeout ، بالطبع لا ، ولكن لعمل ذلك سنقوم بوضع القيمة الثابتة FOREVER والتابعة للصنف Alert أي سنقوم بكتابة الكود على الشكل:

CODE

```
alert.setTimeout(Alert.FOREVER);
```

كما سنرى ذلك في المثال التالي:

CODE

```
public AlertType setType()
```

يقوم بتغيير الشكل الذي سيظهر به التنبيه هل هو ALARM أو CONFIRMATION أو ERROR أو INFO أو WARNING .

والآن بعد شرحنا كيفية التعامل مع التنبيه بقي علينا آن نقوم بوضع مثال يوضح لنا كل هذا :

CODE

```
import javax.microedition.midlet.*;
import javax.microedition.lcdui.*;
```

```
public class MyAlert extends MIDlet implements
CommandListener
{
 private Display display;

Command cmdExit = new Command("Exit",
Command.EXIT, 0);
Command cmdAlert = new Command("Alert",
Command.SCREEN, 0);
Command cmdAlertError = new Command("Error",
Command.SCREEN, 0);
Command cmdAlertSave = new Command("Save",
Command.SCREEN, 0);

  Form form;

  Alert alert;

 public MyAlert  ()
  {
   display =Display.getDisplay(this);
   form = new Form("Form Title");
   alert = new Alert (" Alert Title ");
  }

 public void startApp()
  {
   form.append("This is My Form");
   form.addCommand(cmdExit);
   form.addCommand(cmdAlert);
   form.addCommand(cmdAlertError);
```

```
  form.addCommand(cmdAlertSave);
  form.setCommandListener(this);
  display.setCurrent(form);
 }

 public void pauseApp(){ }
 public void destroyApp(boolean unconditional){}

/************** implements CommandListener
****************/
public void commandAction(Command c,
Displayable s)
{
  if (  s == form )
    {
      if (c == cmdExit)
        {
          destroyApp(false);
          notifyDestroyed();
        }
      else if (c == cmdAlert)
        {
          alert.setString("Alert with command
Ignore\n _____\n Alarm Type ");
          alert.setType(AlertType.ALARM);
          alert.setTimeout(Alert.FOREVER);
          display.setCurrent(alert);
        }
      else if (c == cmdAlertError)
        {
          alert.setString("Alert with 5 second
waitting \n _____\n Error Type");
```

```
            alert.setType(AlertType.ERROR);
            alert.setTimeout(5000);
            display.setCurrent(alert);
          }
        else if (c == cmdAlertSave)
          {
            alert.setString("Alert with 3 second
waitting \n _____\n Confirmation Type");
            alert.setType(AlertType.CONFIRMATION);
            alert.setTimeout(3000);
            display.setCurrent(alert);
          }
      }

  }  // end commandAction
} // end Class
```

CODE

```
Command cmdAlert = new Command("Alert",
Command.SCREEN, 0);
Command cmdAlertError = new Command("Error",
Command.SCREEN, 0);
Command cmdAlertSave = new Command("Save",
Command.SCREEN, 0);
```

عرفنا ثلاث أزرار وسنستخدمها لإظهار الأنواع المختلفة للتنبيه.

CODE

```
 Form form;
```

سنستخدمه من أجل إضافة الأزرار إليه وعرضه فقط.

CODE

```
Alert alert;
```

التنبيه الذي سنقوم بإظهاره على الشاشة مع تغيير لشكل العرض.

CODE

```
public MyAlert   ()
 {
```

هذه المرة سنضع هذه التعليمات الابتدائية في المشيد.

CODE

```
alert = new Alert (" Alert Title ");
```

هنا قمنا بإنشاء كائن من النوع تنبيه وأعطيناه العنوان Alert Title.

CODE

```
form.append("This is My Form");
```

إضافة نص إلى الفورم وسوف ترى هذا النص ظاهراً على الفورم كما بالشكل:

CODE

```
form.addCommand(cmdExit);
form.addCommand(cmdAlert);
form.addCommand(cmdAlertError);
form.addCommand(cmdAlertSave);
```

إضافة الأزرار إلى الفورم.

CODE

```
        else if (c == cmdAlert)
```

إذا قمنا باختيار هذا الزر.

CODE

```
            alert.setString("Alert with command
Ignore\n _____\n Alarm Type ");
```

عرض هذا النص في كائن التنبيه alert.

CODE

```
            alert.setType(AlertType.ALARM);
```

ضبط نوع هذا التنبيه من النوع Alarm كما ترى في هذه الشاشة.

CODE

```
            alert.setTimeout(Alert.FOREVER);
```

سيبقى هذا التنبيه معروضاً على الشاشة إلى أن يقوم المستخدم بكبس الزر done أو ignore.

CODE

```
            display.setCurrent(alert);
```

أخيراً بعد أن أعطينا هذه الخصائص للكائن alert سنقوم هنا بعرضه على الشاشة.

CODE

```
else if (c == cmdAlertError)
```

إذا قمنا باختيار هذا الزر.

CODE

```
alert.setString("Alert with 5 second
waitting \n _____\n Error Type");
```

عرض هذا النص في كائن التنبيه alert.

CODE

```
alert.setType(AlertType.ERROR);
```

ضبط نوع هذا التنبيه من النوع Error كما ترى في هذه الشاشة.

CODE

```
alert.setTimeout(5000);
```

سيبقى هذا التنبيه معروضاً على الشاشة لمدة خمس ثوان فقط ومن ثم يختفي لتظهر الشاشة التي كانت قبله وفي مثالنا هذا ستظهر الفورم Form.

CODE

```
display.setCurrent(alert);
```

أخيراً بعد أن أعطينا هذه الخصائص للكائن alert سنقوم هنا بعرضه على الشاشة.

CODE

```
        else if (c == cmdAlertSave)
```

إذا قمنا باختيار هذا الزر.

CODE

```
            alert.setString("Alert with 3 second
waitting \n _____\n Confirmation Type");
```

عرض هذا النص في كائن التنبيه alert.

CODE

```
            alert.setType(AlertType.CONFIRMATION);
```

ضبط نوع هذا التنبيه من النوع Confirmation وهذا النوع شبيه بتنبيه "تم الحفظ".

CODE

```
            alert.setTimeout(3000);
```

سيبقى هذا التنبيه معروضاً على الشاشة لمدة ثلاث ثوان فقط ومن ثم يختفي.

CODE

```
            display.setCurrent(alert);
```

أخيراً بعد أن أعطينا هذه الخصائص للكائن alert سنقوم هنا بعرضه على الشاشة.

المثال بعد التنفيذ



الفصل الثامن

التعامل مع القوائم List في J2ME، ضمن High Level APIs

سنتعرف في هذا الجزء على كيفية التعامل مع القوائم List.

يرث هذا الصنف عدد من الأعضاء Members من الصنف Screen ومن هذه الطرق setTicker, setTitle كما أنه يستخدم الواجهة Choice على الشكل Choice implements.

List: عبارة عن شاشة تحتوي على مجموعة خيارات Choices يستطيع المستخدم اختيار أي واحد منها.

وهناك ثلاثة أشكال لهذه القائمة:

IMPLICIT: قائمة من الأزرار يستطيع المستخدم الضغط على أي زر من القائمة ويتم معرفة الزر الذي تم كبسه باستخدام الطريقة getSelectedIndex التابعة للكائن المنشأ من الصنف List.

أما SELECT_COMMAND: فيعبر عن الزر اختيار select الذي يأتي مترافقاً مع هذا النوع من القوائم.

EXCLUSIVE: تمثل قائمة خيارات يستطيع المستخدم اختيار خيار واحد فقط من هذه القائمة وتكون قيمة الخيار الذي تم اختياره true وبقية الخيارات false وهذه القائمة شبيهة بال Option في الفيجوال بيسك ويمكن معرفة قيمة كل خيار هل تم اختياره أم لا بواسطة الطريقة isSelected حيث نعطيها رقم الخيار في القائمة وتعيد لنا قيمة true أو false أي هل تم اختياره أم لا.

MULTIPLE: قائمة خيارات من النوع المتعدد وتشبه إلى حد ما CheckBox في الفيجوال بيسك ويتم التعامل معها بنفس التعامل مع النوع EXCLUSIVE.

لنأخذ مثالاً أنه يوجد لدينا قائمة من النوع الأول IMPLICIT فإننا سنعرف أن المستخدم قام بالضغط على زر معين كما في هذا المثال:

CODE

```
public void commandAction (Command c, Displayable d) {
    if (d == myList) {
        if (List.SELECT_COMMAND ==  ) {
            // do Something
        }
    }
}
```

يجب ملاحظة أن كل هذه القوائم يتم إنشاؤها بنفس الطريقة ولكن فقط يختلف النوع.

هذا الكائن يحتوي على مشيدين Constructor تستطيع اختيار أحدهما.

CODE

```
List(String title, int listType)
List(String title, int listType, String[] stringElements, Image[] imageElements)
```

title: العنوان الذي سيظهر أعلى الصندوق.

listType: نوع هذه القائمة وسيترتب على النوع شكل العرض.

وستكتب على الشكل TYPE List حيث أن TYPE ستمثل إما EXCLUSIVE, IMPLICIT, MULTIPLE.

stringElements: مصفوفة نصوص الاختيارات التي ستظهر كعناصر على القائمة.

imageElements: مصفوفة الأيقونات أو الصور التي ستظهر مترافقة مع العناصر.

وإذا لم نرد استخدام أي أيقونه سنضع القيمة null بدلاً عن المصفوفة.

وبعـد أن تعرفنـا علـى بنيـة المـشيد سـنذكر الآن بعـض الطـرق التابعـة للـصنف List وكيفية استخدامها.

CODE

```
public void addCommand(Command cmd)
```

يرث الصنف List هذه الطريقة من الصنف Displayable.

CODE

```
public int append(String stringPart, Image
imagePart)
```

تقوم هذه الطريقة بإضافة عنصر ما إلى القائمة وتعيد رقم العنصر في القائمة.

stringPart: نص العنصر الذي نقوم بإضافته.

imagePart: الأيقونة التي ستظهر مترافقة مع العنصر المضاف.

وإذا أردنا عدم إظهار أي أيقونة سنستخدم القيمة null.

CODE

```
public int getSelectedIndex()
```

تعيد هذه الدالة رقم العنصر الذي تم اختياره حالياً.

CODE

```
public boolean isSelected(int elementNum)
```

تعيد هذه الدالة قيمة true إذا كان هذا العنصر قد تم اختياره أو false إذا لم يتم اختياره.

elementNum: تعبر عن رقم العنصر.

والآن بعد شرحنا كيفية التعامل مع القائمة List بقي علينا أن نقوم بوضع مثال يوضح لنا كل هذا.

CODE

```
import javax.microedition.midlet.*;
import javax.microedition.lcdui.*;

public class MyList extends MIDlet implements
CommandListener
{
 private Display display;

Command cmdExit = new Command("Exit",
Command.EXIT, 0);
Command cmdBack = new Command("Back",
Command.BACK, 0);
Command cmdInfo = new Command("Info",
Command.SCREEN, 0);

  List mainList;
  List subList;

  String[] choices;

  Ticker ticker;

 public MyList  ()
  {
   display =Display.getDisplay(this);
   mainList = new List("Main List",List.IMPLICIT);
```

```
  ticker = new Ticker (" www.arabteam2000.com
");
  choices = new String[3];
  choices[0] = "choice1";
  choices[1] = "choice2";
  choices[2] = "choice3";
  }

 public void startApp()
  {
   mainList.append("EXCLUSIVE",null);
   mainList.append("MULTIPLE",null);
   mainList.addCommand(cmdExit);
   mainList.setTicker(ticker);
   mainList.setCommandListener(this);
   display.setCurrent(mainList);
 }

 private void whatSelected( List list )
  {
    String selected = new String();
    for ( int i = 0; i < list.size(); i++ )
     if ( list.isSelected( i ) )
       selected = selected + " , " + list.getString( i
);
    Alert alert = new Alert("choices");
    if ( selected.length() > 0 )
       alert.setString(" you selected : " + selected);
    else
       alert.setString("you didn`t select");
    display.setCurrent(alert);
  }
```

```
 public void pauseApp(){ }
 public void destroyApp(boolean unconditional){}

/************** implements CommandListener
*****************/
public void commandAction(Command c,
Displayable s)
{
   if (  s == mainList )
     {
       if (c == cmdExit)
         {
           destroyApp(false);
           notifyDestroyed();
         }
      else if ( c == List.SELECT_COMMAND )
        if ( mainList.getSelectedIndex() == 0 )
         {
           subList = new List("EXCLUSIVE List",
List.EXCLUSIVE, choices, null);
           subList.addCommand(cmdBack);
           subList.addCommand(cmdInfo);
           subList.setCommandListener(this);
           display.setCurrent(subList);
         }
        else if (mainList.getSelectedIndex() == 1)
         {
           subList = new List("MULTIPLE
List",List.MULTIPLE, choices, null);
           subList.addCommand(cmdBack);
           subList.addCommand(cmdInfo);
```

```
            subList.setCommandListener(this);
            display.setCurrent(subList);
          }
      }
    else if ( c == cmdInfo )
      whatSelected( (List) s );
    else if ( c == cmdBack )
      {
        mainList.setCommandListener(this);
         display.setCurrent(mainList);
      }
 }  // end commandAction
}  // end Class
```

الفصل التاسع

إضافة صورة Image إلى الفورم Form في J2ME، ضمن High Level

تعرفنا سابقاً على كيفية التعامل مع الفورم Form وكيفية إضافة كائن من النوع TextField إلى الفورم.

سنتعرف الآن على الصنف Image وكيفية إضافتها إلى الفورم.

قبل هذا يجب علينا معرفة ثلاث أشياء:

١ - يجب علينا أولاً معرفة كيفية إنشاء صورة:

لإنشاء صورة وربطها مع التطبيق يجب علينا استخدام الطريقة cteateImage التابعة للصنف Image ، سيكون الكود المستخدم كالآتي:

CODE

```
  try{
          Image img = Image.createImage( /* اسم ملف الصورة مع الدليل */ );
      } catch (java.io.IOException e)
          { /* إظهار رسالة خطأ */ }
```

وسنلاحظ هنا أننا استخدمنا الاستثناءات Exception ومعالجة الأخطاء try و catch.

والسبب في ذلك أن التطبيق يقوم بالبحث عن الصورة في وقت التنفيذ فإذا لم يحصل عليها فإن التطبيق يفشل إذا لم نقم بمعالجة الخطأ.

٢ - يجب علينا معرفة الأنماط التي يمكن عرضها في تطبيق J2ME:

إلى الآن يوجد نوع واحد فقط من الصور التي يمكن إضافتها إلى التطبيق هذا النوع هو:

CODE

```
PNG Image FormatPNG (Portable Network
Graphics)
```

٣ - كيفية إضافة الصورة إلى الحزمة JAR باستخدام Wireless ToolKits .

يمكنك استخدام هذا الأمر:

CODE

```
*/\bin\jar.exe umf MANIFEST.MF دليل الجافا/*
MyApp.jar Image.png
```

يجب الانتباه هنا أننا استخدمنا العبارة umf.

أيضاً يستحسن أن تكون الصورة موجودة في نفس الدليل. أما إذا وضعتها في دليل آخر فإنه يجب كتابة اسم الدليل كاملاً مع ملاحظة أنه سيتم إنشاء نفس الدليل في الكلاسات المحزومة JAR.

لا يوجد أي مشيد Constructor لصنف Image مباشرة بعد تعريف كائن من نوع Image يتم استخدام createImage والسبب أن عملية إنشاء كائن Image يحدث تلقائياً عند استخدام الطريقة createImage

هناك عدة طرق لاستخدام الطريقة createImage .

CODE

```
public static Image createImage(byte[]
imageData, int imageOffset, int imageLength)
```

imageData: مصفوفة تحتوي على بيانات النقاط للصورة على شكل بايتات Bytes.

مثال على ذلك عند تحويل كائن من نوع صورة إلى String باستخدام الدالة toString ومن ثم تحويلها إلى Bytes باستخدام الدالة getBytes

CODE

```
public static Image createImage(Image source)
```

source: كائن صورة موجود من قبل نريد نسخه إلى كائن آخر باستخدام هذه الطريقة.

أي أن هذه الطريقة تستخدم فقط لنسخ كائن من نوع Image إلى كائن آخر من نفس النوع.

CODE

```
public static Image createImage(int width, int
height)
```

إنشاء مساحة فارغة من نمط رسومي لأجل إضافة أي رسومات أو صور لهذه المساحة:

height: ارتفاع هذه المساحة.

width: عرض هذه المساحة.

CODE

```
public static Image createImage(String name)
```

إنشاء كائن من صورة موجودة على شكل ملف محزوم مع الكلاسات المحزومة في الملف jar.

ويجب علينا كتابة اسم الملف مع الدليل الموجود فيه ملف الصورة وهذه الطريقة هي المستخدمة بكثرة.

والآن سنضع مثالاً يوضح كل هذا:

CODE

```
import javax.microedition.lcdui.*;
import javax.microedition.midlet.*;

public class MyForm extends MIDlet  implements
CommandListener
 {
  Display display;
  Form frm;
  Command exitCommand;

 public MyForm()
  {
    frm = new Form("Form Title");
    display = Display.getDisplay(this);
    exitCommand = new Command("Exit",
Command.EXIT, 1);
  }
 public void startApp()
  {
    frm.append("This text in Form");
    addImageToForm();
    frm.append("\nwww.arabteam2000.com");
    frm.addCommand(exitCommand);
    frm.setCommandListener(this);
    display.setCurrent(frm);
  }

  public void  pauseApp() {  }
```

```
 public void destroyApp(boolean unconditional)  {
}

public void addImageToForm()
 {
   try{
        Image img = Image.createImage(
 اسم                                    "/Test.png");  //
ملف الصورة مع الدليل
        frm.append(img);
      } catch (java.io.IOException e)
        {
          Alert alert = new Alert ( "Error: Load
Image" );
          alert.setString( e.getMessage() );
          display.setCurrent(alert);
         }
 }

  public void commandAction(Command c,
Displayable s)
      {
if (c == exitCommand)
          {
 destroyApp(false);
 notifyDestroyed();
}
       }
}
```

الفصل العاشر

إضافة العناصر Items إلى الفورم Form في J2ME، آخر درس ضمن High Level APIs APIs

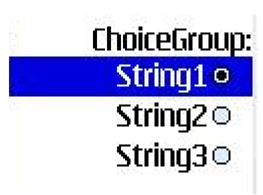
تعرفنا سابقاً على كيفية التعامل مع الفورم Form وكيفية إضافة كائن من النوع TextField إلى الفورم.

سنتعرف الآن على بعض الأصناف الأخرى الشبيهة بالصنف TextField من حيث إضافتها إلى الفورم:

Items: هي الأصناف الأبناء المشتقة من الصنف الأب Item وهذه الأصناف يمكن إضافتها إلى كائن من نوع فورم Form أو من نوع تنبيه Alert ونحن في هذا الشرح سنوضح كيفية إضافتها إلى الكائن Form فقط وسنهمل إضافتها إلى التنبيه Alert وذلك لأن العمليتين متشابهتين.

الأصناف المشتقة من الصنف Item هي DateField, ChoiceGroup, TextField, StringItem, ImageItem, Gauge. ترث هذه الأصناف طريقتين Methods من الصنف Item هما setLabel , getLabel.

ChoiceGroup: عبارة عن قائمة تحتوي على عدد من الاختيارات (التي سنقوم بإضافتها) يستطيع المستخدم اختيار أحد الخيارات أو بعض منها حسب نوعية ChoiceGroup التي سنقوم بتحديدها، وهذا الصنف شبيه بالقائمة List التي شرحناها في سابقاً:



DateField: حقل الوقت والتاريخ وهو عبارة عن صندوق يحتوي على التاريخ أو الوقت أو الاثنين معاً حسب ما تريده.

DateField:
[0000/01/01]

Gauge: المقياس أو المعيار وهو شبيه إلى حد ما بشريط التقدم Progress Bar أو Slider يستطيع المستخدم بزيادة القيمة أو إنقاصها.



ImageItem: هذا الصنف لا يمثل الصورة نفسها وإنما يمثل الموضع الذي ستوضع فيه الصورة ويمكن تشبيهه بالبرواز أو الإطار للصورة حيث يتحكم هذا الصنف بطريقة عرض الصورة مثلاً محاذاة لليمين أو لليسار أو الوسط LAYOUT_CENTER و LAYOUT_LEFT و LAYOUT_RIGHT.

StringItem : عنصر يقوم بعرض نص معين للقراءة فقط غير قابل للتعديل يشبه تماماً عملية إلحاق نص إلى الفورم عبر الطريقة.

CODE

```
public int append(String str)
```

المشروحة في التعامل مع الفورم.

TextField: عنصر صندوق نص يقوم بعرض جزء من النص وإخفاء بقية النص يشبه إلى حد ما عنصر صندوق النص TextBox وقد تعاملنا معه سابقاً في فصل التعامل مع الفورم.

TextField
ABC

ChoiceGroup :-

المشيد Constructor الخاص بهذا الصنف يكون على أحد الشكلين

CODE

```
ChoiceGroup(String label, int choiceType)
ChoiceGroup(String label, int choiceType, String[]
stringElements, Image[] imageElements)
```

label: العنوان الذي سيكتب في الأعلى قبل كتابة الخيارات.

choiceType: النوع ويتم استخدامه بنفس الطريقة المستخدمة في القائمة List بكتابة اسم الصنف ChoiceGroup ثم نقطة ونوع القائمة كالتالي:

ChoiceGroup.TYPE حيث أن TYPE يمثل أحد الأنواع التالية:

IMPLICIT, MULTIPLE EXCLUSIVE

وقد تم توضيح هذه الأنواع عند شرح القائمة List.

stringElements: مصفوفة تحتوي على الخيارات التي تود إضافتها إلى ChoiceGroup.

imageElements: مصفوفة الأيقونات التي نرغب بإضافتها مع الخيارات وسيظهر الخيار مع صورة الأيقونة وإذا أردنا عدم استخدام أي أيقونة نقوم بوضع القيمة null بدلاً عن المصفوفة.

معظم طرق هذا الصنف شبيهة بالطرق التابعة للصنف List لذا لن نقوم بشرحها وسنقوم فيما بعد بإضافة مثال يوضح كيفية إضافة كائن من هذا الصنف إلى كائن من نوع Form

CODE

```
import javax.microedition.lcdui.*;
import javax.microedition.midlet.*;

public class MyForm extends MIDlet  implements
```

```
CommandListener
 {
  Display display;  // The display for this MIDlet
  Form frm;                          // The Window
  Command exitCommand; // The exit command

 public MyForm()
  {
    frm = new Form("Form Title");
    display = Display.getDisplay(this);
    exitCommand = new Command("Exit",
Command.EXIT, 1);
  }
 public void startApp()
  {
   addChoiceGroupToForm();
//   إضافة زر الخروج إلى الفورم
    frm.addCommand(exitCommand);
//   تشغيل المتصنت للاستجابة لأوامر الفورم
    frm.setCommandListener(this);
//   عرض النافذة على شاشة التطبيق
    display.setCurrent(frm);
  }

  public void  pauseApp() {  }

  public void destroyApp(boolean unconditional)  {
}

 public void addChoiceGroupToForm()
  {
```

```
// إضافة نص إلى بداية الفورم
        frm.append("This Text in Form");
        ChoiceGroup choice = new
ChoiceGroup("Choice Title",
                                Choice.EXCLUSIVE);  //
اختيار واحد من عدة خيارات
// كما يوجد النوعان
        // Choice.MULTIPLE , Choice.IMPLICIT
// إضافة الاختيارات للقائمة
        choice.append("String1",null);
        choice.append("String2",null);
        choice.append("String3",null);
// إضافة القائمة إلى الفورم
        frm.append(choice);
  }

   public void commandAction(Command c,
Displayable s)
       {
if (c == exitCommand)
           {
 destroyApp(false);
 notifyDestroyed();
}
        }
}
```

:Gauge

المشيد Constructor الخاص بهذا الصنف يكون على الشكل:

CODE

```
Gauge(String label, boolean interactive, int
maxValue, int initialValue)
```

label: العنوان أو النص الذي يظهر أعلى Gauge.

interactive: يحدد إذا كان المستخدم يستطيع تغيير القيمة أم لا وذلك بوضع القيمة true أو false.

maxValue: أعلى قيمة ممكن أن يصل إليها قيمة Gauge وتبدأ القيم من صفر إلى أعلى قيمة.

initialValue: القيمة الابتدائية التي سيظهر بها Gauge ويجب أن تكون بين الصفر وأعلى قيمة maxValue.

الآن سنوضح بعض الطرق الخاصة بالصنف Gauge.

CODE

```
public int getMaxValue()
```

تعيد هذه الدالة أعلى قيمة ممكن أن تصل إليها قيمة Gauge أو بمعنى آخر فهي تعيد القيمة maxValue التي قمت بضبطها سابقاً

CODE

```
public int getValue()
```

تعيد هذه الدالة القيمة الحالية لـ Gauge.

CODE

```
public boolean isInteractive()
```

من اسم هذه الدالة وطريقة تركيبها نستطيع القول أن هذه الدالة تعيد القيمة true إذا كان المستخدم يستطيع تغيير قيمة Gauge وتعيد false إذا كان لا يستطيع تغيير القيمة

CODE

```
public void setMaxValue(int maxValue)
```

نستطيع ضبط أعلى قيمة لل Gauge من خلال هذه الدالة.

CODE

```
public void setValue(int value)
```

كذلك نستطيع تغيير قيمة Gauge من خلال هذه الطريقة.

الآن سنضع كود يوضح عمل Gauge.

CODE

```
import javax.microedition.lcdui.*;
import javax.microedition.midlet.*;

public class MyGauge extends MIDlet  implements
CommandListener
 {
  Display display;
```

```
  Form frm;
  Gauge gauge;
  Command cmdExit =  new Command("Exit",
Command.EXIT, 0);
  Command cmdGetValue =  new Command("get
value", Command.SCREEN, 0);

 public MyGauge()
  {
    frm = new Form("Form Title");
    display = Display.getDisplay(this);
    gauge = new Gauge("Progress", false, 10, 0);
  }
 public void startApp()
  {
    frm.append("Please wait...");
    frm.addCommand(cmdExit);
    frm.setCommandListener(this);
    display.setCurrent(frm);
    addGaugeToForm();
    gauge = new Gauge("Gauge", true, 30, 10);
    frm.append("..................\n");
    frm.append( gauge );
    frm.addCommand(cmdGetValue);
  }

  public void  pauseApp() {  }

  public void destroyApp(boolean unconditional)  {
}

//إذا كنت تستخدم
```

```
// Wireless Toolkits
//لن يظهر لديك عمل هذا الإجراء
//يقوم هذا الإجراء بإظهار شريط التقدم يسير من الصفر حتى أعلى قيمة
//يظهر عمل هذا الإجراء عند استخدام محاكي الموبايل لنوكيا
//أو عند تحميل هذا البرنامج مباشرة على الموبايل
 public void addGaugeToForm()
  {
    Thread thread = new Thread();
    frm.append( gauge );
    for (int i=0; i<gauge.getMaxValue(); i++)
     {
       int value = gauge.getValue();
       try  { thread.sleep(300); }
       catch(InterruptedException e){}
       gauge.setValue( value+1 );
     }
  }

   public void commandAction(Command c,
Displayable s)
     {
       if ( s == frm)
          {
           if (c == cmdExit)
            {
  destroyApp(false);
  notifyDestroyed();
}
           else if ( c == cmdGetValue )
            {
             Alert alert = new Alert ( "My Gauge" );
```

```
            alert.setString( "gauge value is : " +
Integer.toString( gauge.getValue() ) );
            display.setCurrent(alert);
           }
       }
    }
}
```

:TextField

المشيد Constructor الخاص بهذا الكائن على الشكل:

CODE

```
TextBox(String title, String text, int maxSize, int constraints)
```

title: عنوان النص وسيظهر في أعلى صندوق النص.

ونستطيع تغييره باستخدام الطريقة الموجودة في نفس الصنف setTitle وهذه الطريقة ورثها هذا الصنف TextBox من الصنف Screen.

text: وهو النص الذي سيكون موجوداً في داخل صندوق النص ونستطيع التعامل معه من خلال الأمرين setString , getString .

maxSize: عدد الأحرف التي سيتقبلها صندوق النص بحيث لا يمكن للمستخدم إضافة أكثر من العدد المطلوب.

constraints: القيد أو الهيئة للنص المدخل وهناك العديد من القيود. هذه القيود موجودة في الصنف TextField على هيئة حقول ساكنه static ويتم استخدامها على الشكل TextField.Field حيث Field تعبر عن اسم القيد من هذه القيود:

ANY: ويعبر عن جميع الأحرف أو الأرقام.

NUMERIC: يسمح للمستخدم باستخدام الأرقام فقط.

PASSWORD: تظهر للمستخدم نجمة عن كل حرف يكتبه.

وبعـد أن تعرفنـا علـى بنيـة المـشيد سـنذكر الآن بعـض الطـرق التابعـة للـصنف TextBox:

CODE

```
Public void delete(int offset, int length)
```

تقوم هذه الدالة بحذف جزء من النص الموجود في صندوق النص.

offset: بداية النص المراد حذفه.

length: طول النص.

CODE

```
public int getConstraints()
```

تعيد نوع القيد المستخدم على شكل رقم.

CODE

```
public void setConstraints(int constraints)
```

تستطيع تغيير القيد المستخدم في الصندوق باستخدام هذا المنهج.

CODE

```
public int getMaxSize()
```

تعيد الحد الأقصى لعدد الحروف الممكن كتابتها في صندوق النص.

CODE

```
public int setMaxSize(int maxSize)
```

أيضاً نستطيع تغيير الحد الأقصى لعدد الحروف باستخدام هذا المنهج.

CODE

```
public String getString()
```

تعيد هذه الدالة النص المكتوب في الصندوق.

CODE

```
public void setString(String text)
```

يقوم هذا المنهج بوضع النص الذي تريده على الصندوق.

CODE

```
public int size()
```

تعيد هذه الدالة عدد الأحرف المكتوبة حالياً في الصندوق.

الفصل الحادي عشر

عمل الجرافيك والكلاس Canvas

كيفية عمل الجرافيك,, والرسم داخل الشاشة,,الخطوط وأنواعها وكذلك الألوان وأتمنى أن يكون الشرح واضحاً.

تستخدم بكثرة لعمل الألعاب واحتوائها على أساليب مثل:

CODE

```
showNotify()
hideNotify()
keyPressed()
keyRepeated()
keyReleased()
pointerPressed()
pointerDragged()
pointerReleased()
paint()
```

أولاً : الألوان ,,

لتحديد اللون لجرافيك معين أو شكل معين نحتاج إلى الأسلوب:

CODE

```
Graphics.setColor(int red,int green,int blue);
```

فمثلاً,,

CODE

```
setColor(255,0,0);   // أحمر
setColor(0,255,0);   // أخضر
setColor(0,0,255);   // أزرق
setColor(0,0,0);   // أسود
```

ولكل لون قيمة تبدأ من ٠ إلى ٢٥٠ وعندما تكون قيم مختلفة فإن الألوان تمزج مع بعضها حسب القيمة المعطاة وعندما يضع الشخص جميع القيم متسأوية مثل:

CODE

```
setColor(255,255,255);// أبيض
```

يمكن استبدال ذلك بالأسلوب:

CODE

```
setGrayScale(255);
```

ثانياً: كيفية عمل الجرافيك,, والرسم داخل الشاشة,,

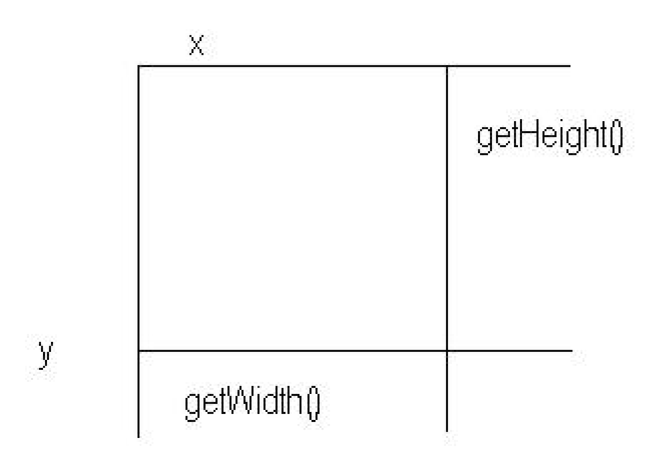
الكلاس Canvas تحتوي على Graphics Object الذي يحتوي على الأسلوب paint-method الذي يمكن عن طريقه رسم الأشكال البسيطة.

أمثلة:

w=Width=العرض

h=Height=الطول

وهذه رسمة توضح إحداثيات الشاشة,



CODE

```
drawRect(int x,int y,int w,int h);//مستطيل
drawRoundRect(int x,int y,int w,int h,int
radius);//مستطيل دائري
drawLine(int x0,int y0,int x1,int y1);//خط بين النقطتين
drawArc(int x,int y,int width,int height,int
startAngle,int ArcAngle);//**
```

❖ يقوم برسم Ellipse ليست مغلقة مثل القوس ,, المقصود بـ startAngle هي زاوية البداية لل Arc of a Circle بالدرجات.

CODE

```
drawArc(110,110,80,80,45,270);
```

فمثلاً هنا زاوية البداية ٤٥ درجة والمقصود بـ arcAngle هو زاوية قوس الدائرة بالدرجات.

في المثال ٢٧٠ درجة الزوايا الموجبة عكس عقارب لساعة والسالبة مع عقارب الساعة وهناك أيضاً أساليب لتعبئة الشكل بدلاً من draw نكتب fill.

CODE

```
fillRect(int x,int y,int w,int h);
fillRoundRect(int x,int y,int w,int h,int radius);
fillArc(int x,int y,int width,int height,int
startAngle,int ArcAngle);
```

وهناك أسلوب يتم عن طريقه عمل الجرافيك أو الشكل المرسوم بشكل متقطع أو غير متقطع عن طريق إضافة Graphics.DOTTED و Graphics.SOLID إلى الأسلوب.

CODE

```
Graphics.setStrokeStyle(Graphics.DOTTED);
Graphics.setStrokeStyle(Graphics.SOLID);
```

وهذا مثال يوضح لنا ما ذكرنا سابقاً مع بعض الملاحظات.

يستحسن عمل الأسلوب paint في كلاس لوحده وكذلك يجب أن يكون الكلاس مشتق من Canvas ونحتاج إلى الحزمة lcdui الموجودة بها الكلاس Canvas.

CODE

```
import javax.microedition.lcdui.*;

public class DrawingDemoCanvas extends Canvas

{
// The paint Method

public void paint(Graphics g)

{
g.setGrayScale(255);
g.fillRect(0,0,getWidth(),getHeight());
g.setGrayScale(0);
g.setStrokeStyle(Graphics.DOTTED);
g.drawLine(0,0,100,200);
g.fillRect(20,30,30,20);
```

```
//.
//.
//.
}
}
```

CODE

```
import javax.microedition.midlet.*;
import javax.microedition.lcdui.*;
public class DrawingDemo extends MIdlet

{

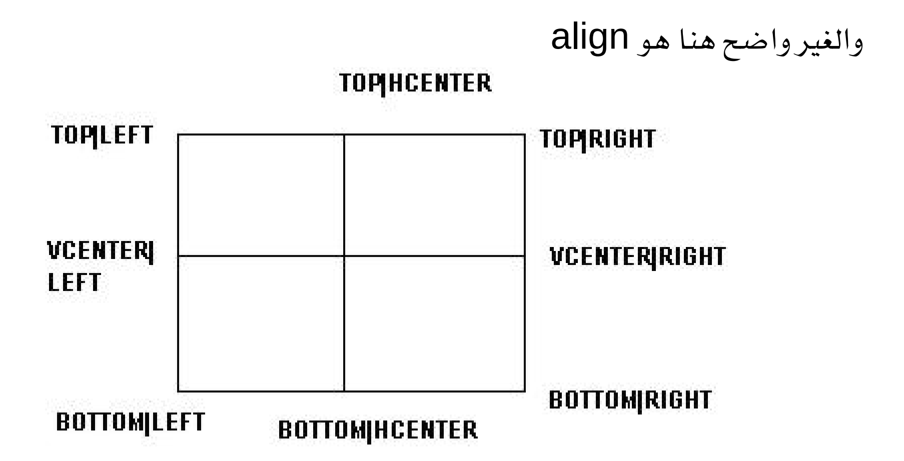
public void startApp()
{
Display.getDisplay(this).setCurrent(new
DrawingDemoCanvas());
}
public void pauseApp(){}
public void destroyApp(boolean forced){}
}
```

سأقوم بشرح أسلوبين وهما يستخدمان ضمن الأسلوب paint وهما:

CODE

```
Graphics.drawImage(Image image,int y,int y,int
align);
Graphics.drawString(String text,int x,int y,int
align);
```

والغير واضح هنا هو align



نلاحظ في الصورة تقسيم الشاشة ما معنى align حيث تتكون من شقين مثل:

CODE

```
public void paint(Graphics g) {
g.drawString("www.arabteam2000-
forum.com",0,0,g.TOP | g.LEFT);
}
```

ثالثاً: الخطوط وأنواعها ,,

أنواع الخطوط:

STYLE_PLAIN,STYLE_ITALIC,STYLE_BOLD,STYLE_UNDERLINED

حجم الخط:

SIZE_SMALL,SIZE_MEDIUM,SIZE_LARGE

مظهر الخط:

FACE_SYSTEM,FACE_MONOSPACE,FACE_PROPORTIONAL

وهذا مثال لاستخدام خط ,,حجمه، نوعه ومظهره,,:

CODE

```
g.setFont(Font.getFont(Font.FACE_SYSTEM,
Font.STYLE_BOLD,Font.SIZE_MEDIUM));
```

حيث يمكن استخدام الآتي واستخدامه مباشرة:

CODE

```
Font font = Font.getFont(FACE_PROPORTIONAL,
STYLE_BOLD, SIZE_LARGE);
```

CODE

```
g.setFont(Font.getFont(Font.FACE_SYSTEM,
Font.STYLE_BOLD, Font.SIZE_MEDIUM));
g.drawString("www.arabteam2000-
forum.com",0,0,g.TOP | g.LEFT);
```

وهذا مثال يوضح ما سبق:

CODE

```
import javax.microedition.midlet.*;
import javax.microedition.lcdui.*;

public class graph1 extends MIDlet {

public graph1() {
}

public void startApp() {

Canvas canvas = new MyCanvas();
Display display = Display.getDisplay(this);
display.setCurrent(canvas);
}
public void pauseApp() {
}
public void destroyApp(boolean unconditional) {
}
}
```

CODE

```
import javax.microedition.lcdui.*;

public class MyCanvas extends Canvas {

public void paint(Graphics g) {

g.setColor(0,0,0);
g.fillRect(0,0,getWidth(), getHeight());
g.setColor(255,255,255);
g.setFont(Font.getFont(Font.FACE_SYSTEM,
Font.STYLE_BOLD,Font.SIZE_MEDIUM));
g.drawString("www.arabteam2000-
forum.com",0,0,g.TOP | g.LEFT);
}
}
```

سأريكم مثال لكيفية استخدام الأحداث ضمن Low-Level Events:

CODE

```
import javax.microedition.lcdui.*;

public class llevent extends MIDlet {
Display display;
Command exit;

public llevent() {
display = Display.getDisplay(this);
```

```
}

public void destroyApp(boolean unconditional) {
}

public void pauseApp() {
System.out.println("App paused.");
}

public void startApp() {
display = Display.getDisplay(this);

Canvas canvas = new Canvas() {

private String message = "Press any key";
private int x1 = (getWidth()/2);
private int y1 = (getHeight()/2);


public void paint(Graphics g)
{


g.setColor( 255, 255, 255 );
g.fillRect( 0, 0, getWidth(), getHeight() );
g.setColor( 0, 0, 0 );
g.drawString( message, getWidth()/2, 0,g.TOP |
g.HCENTER );

g.drawLine(x1,y1-5,x1,y1+5);
g.drawLine(x1-5,y1,x1+5,y1);
```

```
}

protected void keyPressed( int keyCode ){
if( keyCode == getKeyCode( FIRE ) )
{
message = "FIRE";
} else if( keyCode == getKeyCode( LEFT ) ){
message = "LEFT"; x1--;
} else if( keyCode ==getKeyCode( RIGHT ) ){
message = "RIGHT";x1++;
} else if( keyCode == getKeyCode( UP ) ){
message = "UP"; y1--;
} else if( keyCode == getKeyCode( DOWN ) ){
message = "DOWN"; y1++;
} else {
message = getKeyName( keyCode );
}


repaint();
}


};

exit = new Command("Exit", Command.STOP,1);
canvas.addCommand(exit);
canvas.setCommandListener(new
CommandListener() {
public void commandAction(Command c,
Displayable d) {
```

```
if(c == exit) {
notifyDestroyed();
}
else {
System.out.println("Saw the command: "+c);
}
}
});
display.setCurrent(canvas);
}
}
```

فإن في هذا المثال يتم استخدام الأزرار (٢)، (٤)، (٦) (٨) حيث عن طريقها يتم تحريك الإشارة + مع إظهار رسالة للاتجاه الذي يتم التحريك إليه.

الفصل الثاني عشر

حفظ البيانات الدائم Package javax.microedition.rms

سنتعرف في هذا الفصل على كيفية حفظ البيانات مثل دليل الهاتف ,, مذكرة وغيرها الكثير من التطبيقات وهو بالفعل فصل صعب أتمنى من الله أن أقوم بتبسيطه لكم .

أولا : الكلاس RecordStore تستخدم هذه الكلاس والأساليب الموجودة فيها لكتابة أو قراءة بيانات وكذلك لتخزينها.

الاسم Record-Store يجب أن يكون عبارة عن String ويجب أن يتم هناك التمييز بين الكتابة الصغيرة والكبيرة فمثلاً "ArabTeam " ليست مثل" arabteam" وهناك أسلوب يفتح Record-Store وينتجه إذا لم يكن موجوداً

CODE

```
RecordStore myRecord =
RecordStore.openRecordStore("myRecord",true);
```

وهذه مجموعة من الأساليب:

CODE

```
RecordStore openRecordStore(String
recordstoreName, boolean creatIfNecessary)
RecordStore closeRecordStore()
void deleteRecordStore(String
 Record-StoreتمحيrecordStoreName)//
```

عندما يقوم الشخص بفتح Record-Store معين مثل:

CODE

```
RecordStore myRecord =
RecordStore.openRecordStore("myRecord",true);
```

يستطيع الشخص إضافة بيانات عن طريق:

CODE

```
int addRecord(byte[] data, int offset, int
numBytes)
```

مثل:

CODE

```
bytes [] myByte ="Hello".getBytes();
myRecord.addRecord(myByte,0,myByte.length);
```

حتى تتم قراءة Records يتم استخدام الأسلوب:

CODE

```
byte[] getRecord(int recordId)
```

وهذه مجموعة من الأساليب:

CODE

```
int getNumRecords()
```

لمعرفة عدد Records في Record Store:

CODE

```
int getName()
```

لمعرفة اسم Record Store:

CODE

```
int getSize()
```

سنقوم بعمل مثال بسيط باستخدام الكلاس RecordStore والأساليب الموجودة فيها وهو عبارة عن دفتر للمذكرات اليومية بحيث يستطيع المستخدم تخزين ما يريد في TextBox والاطلاع عليه متى يريد لعمل ذلك التطبيق نحتاج إلى TextBox وكذلك الأزرار التي نستطيع إلى الآن بكل سهولة إنشائها ويبقى علينا هو تخزين ما تم كتابته باستخدام RecordStore والأساليب الموجودة فيها نقوم بإنشاء RecordStore ونسميها diary ونحتاج أيضاً إلى ID (رقم المدخل أو الصفحة) تابع المدخل الحالي ونقوم بعمل ذلك عن طريق:

CODE

```
RecordStore diary;
int currentId=1;
```

CODE

```
try {
diary = RecordStore.openRecordStore("diary",
true);
String text =loadEntry(diary.getNumRecords ());
textBox.setString(text);
textBox.setTitle("Diary - Day " + currentId);
}
catch (RecordStoreException e) {
throw new RuntimeException("Cannot open diary;
reason: "+e);
}
```

سيقوم هنا بفتح RecordStore واسمها diary وفي حالة عدم القدرة على ذلك يعطينا السبب.

وهناك مجموعة من الاستثناءات ومنها:

CODE

```
InvalidRecordIDException(String message)
InvalidRecordIDException()
RecordStoreException(String message)
```

```
RecordStoreException()
RecordStoreFullException()
RecordStoreFullException(String message)
RecordStoreNotFoundException()
RecordStoreNotFoundException(String message)
RecordStoreNotOpenException()
RecordStoreNotOpenException(String message)
```

وما تقوم به هو ترجمة ال Exception إلى العربية مثل:

CODE

```
RecordStoreNotFoundException()
```

عندما يتم إيجادRecordStore المعطاة عند استخدام الأساليب:

CODE

```
openRecordStore()
و
deleteRecordStore()
```

ونقوم بعمل أسلوب آخر وهو loadEntry()-Method الذي يقوم بشحن المدخل من RecordStore وإعطاء المدخل رقم وفي حال تواجد الرقم يعطى رقم جديد للصفحة أقصد بالمدخل وال currentId هو ال TextBox ورقمه أي الصفحة ورقمها عند تنفيذ البرنامج يظهر عندTitle تابع TextBox فوق diary + رقم

المدخل ويكون الناتج عبارة عن String الذي سيتم تخزينه وهذا يقوم فقط بشحن المدخلات وليس تخزينها.

CODE

```
public String loadEntry (int newId) throws
RecordStoreException {
if (newId < 1 || newId > diary.getNumRecords ())
{
byte [] data = " ".getBytes ();
currentId = diary.addRecord(data, 0, data.length);
}
else
currentId = newId;
return new String
(diary.getRecord(currentId)).trim ();
}
```

trim() لترتيب ال Record.

ولتخزين ما تم شحنه نحتاج إلى saveEntry()-Method التي تؤخذ من رقم الصفحة مثل ما تم كتابته بالصفحة الأولى يتم تخزينه وهكذا حيث يتم تخزين ما كتب في TextBox ك String.

CODE

```
public void saveEntry (String entry) throws
RecordStoreException {
```

```
byte [] data = (entry + " ").getBytes ();
diary.setRecord (currentId, data, 0, data.length);
}
```

وحتـــى لا تـــتم خـــسارة مـــا تم حفظـــه نحتـــاج إلى عمـــل الآتـــي في الأســـلوب destroyApp()-Method:

CODE

```
public void destroyApp (boolean unconditional) {
try {
String text;
text = textBox.getString ();
saveEntry (text);
diary.closeRecordStore ();
}
catch (RecordStoreException e) {
throw new RuntimeException
("Cannot close Diary; reason: "+e);
}
}
```

الفصل الثالث عشر

كيفية عمل Animation في J2ME، شرح لمثال بسيط

كيفية عمل Animation في ,, J2ME

سأقوم بشرح مثال لكيفية عمل رسومات متحركة باستخدام J2ME وهو عبارة عن دائرة كاملة ملونة ويختفي اللون بشكل تدريجي إلى أن تصبح الدائرة باللون الأبيض مثل الساعة تماماً.

لعمل ذلك نحتاج إلى الحزمة:

CODE

```
import javax.microedition.lcdui.*;
```

CODE

```
public class StopWatchCanvas extends Canvas
implements Runnable{
```

سنستخدم الأسلوب paint()-Method لذلك تكون الميدليت مشتقة من Canvas و Runnable حتى نستطيع استخدام

CODE

```
Display display;
display.callSerially(this);
```

وهذا الأسلوب حتى نستطيع القيام بإضافة حدث معين مثل حركة أو الأفضل أن نضيف Thread.

حتى نستطيع إعادة تنفيذ الأوامر الموجودة في paint لابد أن نستخدم أسلوب آخر ونسميه run ونكتب فيه repaint وكيفية البدء في الرسم وإعادة ذلك وهنا يتم حساب رقم الدرجة (الزأوية بالدرجات يعني رسم drawArc في الدائرة قوس قوس) الحالي ومن ثم إعادة الرسم paint عن طريق repaint

CODE

```
public void run()

{

 int permille=(int)((System.currentTimeMillis()-
startTime)/seconds);
 degree = 360 -(permille*360)/1000;
 repaint();
}
```

وهذا هو الكلاس المهم في البرنامج وخير طريقة للفهم السؤال:

CODE

```
import javax.microedition.lcdui.*;

public class StopWatchCanvas extends Canvas
implements Runnable{
```

```
int degree = 360;
long startTime;
int seconds;
Display display;

public StopWatchCanvas ( Display display, int
seconds)

{
this.display =display;
this.seconds = seconds;
startTime =System.currentTimeMillis();
}
public void paint (Graphics g)

{
g.setGrayScale(255);
g.fillRect(0,0,getWidth(),getHeight());
if(degree>0){
                g.setColor(255,0,0);

g.fillArc(0,0,getWidth(),getHeight(),90,degree);
                display.callSerially(this);
                }
                g.setGrayScale(0);
                g.drawArc(0,0,getWidth()-
1,getHeight()-1,0,360);

}

public void run()
```

```
{

 int permille=(int)((System.currentTimeMillis()-
startTime)/seconds);
 degree = 360 -(permille*360)/1000;
 repaint();
}

}
```

الفصل الرابع عشر

Package javax.microedition.media، شرح لهذه الحزمة

هذه الحزمة تستخدم بشكل عام من أجل تشغيل الرنات أو تدعم تشغيل الرنات مثل عند ضغط زر معين حدوث رنة أو شئ من هذا القبيل، وهذه الحزمة تحتوي على كلاس وهي Manager التي تدعم تشغيل الرنات باستخدام الـ Interface Player.

والكلاس Player هي التي تستخدم بكثرة مع Manager حيث يتم تشغيل الرنات من format مثل midi,wav or mp3 وهذا الكود لتشغيل رنة من الإنترنت أو اسم الدليل.

CODE

```
try{
Player p=Manager.createPlayer("  ****  ");
p.start();
}
catch(MediaException pe){}
catch(IOException pe  ){}
```

اسم الدليل الموجود فيه النغمة مثل naghm.mid/ .

ويتم التشغيل عن طريق p.start()-Method وهذا يستخدم بكثرة.

وهناك أيضاً الحزمة javax.microedition.media.control Package التي يتم عن طريقها التحكم بصفات و سرعة الصوت أو درجة الصوت مثل عالي منخفض.

برامج للتنفيذ

تحويل التاريخ الميلادي للهجري

CODE

```
import javax.microedition.lcdui.*;
import javax.microedition.midlet.*;
import java.io.*;

public class ChangeTo extends MIDlet implements
CommandListener
{
Display display;

Form fr;
TextField day,year,month;
Command ok = new Command("Ok",Command.OK,1);
Command e  = new Command("Exit",Command.EXIT,0);
Alert result;
public ChangeTo()
{
display  = Display.getDisplay(this);

}

public void startApp()
{
fr       = new Form("ChangeToHijri");
day      = new TextField("Day","",10,TextField.NUMERIC);
month    = new
TextField("Month","",10,TextField.NUMERIC);
year     = new TextField("Year","",
10,TextField.NUMERIC);
result   = new Alert("The Result");
```

```
fr.append(day);
fr.append(month);
fr.append(year);
fr.addCommand(ok);
fr.addCommand(e);
fr.setCommandListener(this);
display.setCurrent(fr);
}

public void pauseApp() {}

public void destroyApp(boolean unconditional) {}

public void commandAction(Command c,Displayable d)
{
if(c == e)
{
	destroyApp(true);
	notifyDestroyed();
}
else if(c == ok)
{
 int da = getNumber(day);
 int ma = getNumber(month);
 int ya = getNumber(year);
 String re = getDate(da,ma,ya);
 result.setString(re );
 result.setTimeout(Alert.FOREVER);
	day.setString("");
 month.setString("");
 year.setString("");
 display.setCurrent(result);
}
}
```

```
public int getNumber(TextField tf)
{
 int n = 0;
 String  s = tf.getString();
  try {
           n = Integer.parseInt(s);
         } catch (NumberFormatException e)
{System.out.println("Please an Integer Number");}
       return n;
}
public String getDate(int d, int m , int y)
{
int jd = 0;
if ( (y > 1582) || ( (y == 1582) && (m > 10)) || ( (y == 1582)
&& (m == 10) && (d > 14))) {

jd = (int)( (1461 * (y + 4800 + ( (m - 14) / 12))) / 4) + (
(367 * (m - 2 - 12 * ( ( (m - 14) / 12)))) / 12) -( (3 * ( ( (y +
4900 + ( (m - 14) / 12)) / 100))) / 4) + d - 32075;
  }
  else {
    jd =(int) (367 * y - ( (7 * (y + 5001 + ( (m - 9) / 7))) / 4)
+( (275 * m) / 9) + d + 1729777);
  }
  int l = jd - 1948440 + 10632;
  int n = ( (l - 1) / 10631);
  l = l - 10631 * n + 354;
  int j = ( ( (10985 - l) / 5316)) * ( ( (50 * l) / 17719)) +( (l /
5670)) * ( ( (43 * l) / 15238));
  l = (int) (l - ( ( (30 - j) / 15)) * ( ( (17719 * j) / 50))-( (j /
16)) * ( ( (15238 * j) / 43)) + 29);
  m = (int) ( ( (24 * l) / 709));
  d = (int) (l - ( (709 * m) / 24));
```

```
  y = (int) (30 * n + j - 30);

return (d + "/" + m + "/" + y);
}
}
```

الفهرس